

```

1 #include <iostream>
2 #include <stdlib.h>
3 #include <graphics.h>
4 #include <math.h>
5 using namespace std;
6 class POLYGON
7 {
8 private:
9     int p[10][10], Trans_result[10][10], Trans_matrix[10][10];
10    float Rotation_result[10][10], Rotation_matrix[10][10];
11    float Scaling_result[10][10], Scaling_matrix[10][10];
12    float Shearing_result[10][10], Shearing_matrix[10][10];
13    int Reflection_result[10][10], Reflection_matrix[10][10];
14
15 public:
16     int accept_poly(int[][10]);
17     void draw_poly(int[][10], int);
18     void draw_polyfloat(float[][10], int);
19     void matmult(int[][10], int[][10], int, int, int, int[][10]);
20     void matmultfloat(float[][10], int[][10], int, int, int, float[][10]);
21     void shearing(int[][10], int);
22     void scaling(int[][10], int);
23     void rotation(int[][10], int);
24     void translation(int[][10], int);
25     void reflection(int[][10], int);
26 };
27 int POLYGON ::accept_poly(int p[][10])
28 {
29     int i, n;
30     cout << "\n\n\t\tEnter no.of vertices:";
31     cin >> n;
32     for (i = 0; i < n; i++)
33     {
34         cout << "\n\n\t\tEnter (x,y)Co-ordinate of point P" << i << ": ";
35         cin >> p[i][0] >> p[i][1];
36         p[i][2] = 1;
37     }
38     for (i = 0; i < n; i++)
39     {
40         cout << "\n";
41         for (int j = 0; j < 3; j++)
42         {
43             cout << p[i][j] << "\t";
44         }
45     }
46     return n;
47 }
48 void POLYGON ::draw_poly(int p[][10], int n)
49 {
50     int i, gd = DETECT, gm;
51     initgraph(&gd, &gm, NULL);
52     line(320, 0, 320, 480);
53     line(0, 240, 640, 240);
54     for (i = 0; i < n; i++)
55     {
56         if (i < n - 1)
57         {
58             line(p[i][0] + 320, -p[i][1] + 240, p[i + 1][0] + 320, -p[i + 1][1] +
240);

```

```

59     }
60     else
61         line(p[i][0] + 320, -p[i][1] + 240, p[0][0] + 320, -p[0][1] + 240);
62     }
63     delay(3000);
64 }
65 void POLYGON ::draw_polyfloat(float p[][10], int n)
66 {
67     int i, gd = DETECT, gm;
68     initgraph(&gd, &gm, NULL);
69     line(320, 0, 320, 480);
70     line(0, 240, 640, 240);
71     for (i = 0; i < n; i++)
72     {
73         if (i < n - 1)
74         {
75             line(p[i][0] + 320, -p[i][1] + 240, p[i + 1][0] + 320, -p[i + 1][1] +
240);
76         }
77         else
78             line(p[i][0] + 320, -p[i][1] + 240, p[0][0] + 320, -p[0][1] + 240);
79     }
80     // delay(8000);
81 }
82 void POLYGON ::translation(int p[10][10], int n)
83 {
84     int tx, ty, i, j;
85     int i1, j1, k1, r1, c1, c2;
86     r1 = n;
87     c1 = c2 = 3;
88     cout << "\n\n\t\tEnter X-Translation tx: ";
89     cin >> tx;
90     cout << "\n\n\t\tEnter Y-Translation ty: ";
91     cin >> ty;
92     for (i = 0; i < 3; i++)
93         for (j = 0; j < 3; j++)
94             Trans_matrix[i][j] = 0;
95     Trans_matrix[0][0] = Trans_matrix[1][1] = Trans_matrix[2][2] = 1;
96     Trans_matrix[2][0] = tx;
97     Trans_matrix[2][1] = ty;
98     for (i1 = 0; i1 < 10; i1++)
99         for (j1 = 0; j1 < 10; j1++)
100             Trans_result[i1][j1] = 0;
101     for (i1 = 0; i1 < r1; i1++)
102         for (j1 = 0; j1 < c2; j1++)
103             for (k1 = 0; k1 < c1; k1++)
104                 Trans_result[i1][j1] = Trans_result[i1][j1] + (p[i1][k1] *
Trans_matrix[k1][j1]);
105     cout << "\n\n\t\tPolygon after Translationâ€¦";
106     draw_poly(Trans_result, n);
107 }
108 void POLYGON ::rotation(int p[][10], int n)
109 {
110     float type, Ang, Sinang, Cosang;
111     int i, j;
112     int i1, j1, k1, r1, c1, c2;
113     r1 = n;
114     c1 = c2 = 3;
115     cout << "\n\n\t\tEnter the angle of rotation in degrees: ";
116     cin >> Ang;

```

```

117     cout << "\n\n **** Rotation Types ****";
118     cout << "\n\n\t\t1.Clockwise Rotation \n\n\t\t2.Anti-Clockwise Rotation ";
119     cout << "\n\n\t\tEnter your choice(1-2): ";
120     cin >> type;
121     Ang = (Ang * 6.2832) / 360;
122     Sinang = sin(Ang);
123     Cosang = cos(Ang);
124     cout << "Mark1";
125     for (i = 0; i < 3; i++)
126         for (j = 0; j < 3; j++)
127             Rotation_matrix[i][j] = 0;
128     cout << "Mark2";
129     Rotation_matrix[0][0] = Rotation_matrix[1][1] = Cosang;
130     Rotation_matrix[0][1] = Rotation_matrix[1][0] = Sinang;
131     Rotation_matrix[2][2] = 1;
132     if (type == 1)
133         Rotation_matrix[0][1] = -Sinang;
134     else
135         Rotation_matrix[1][0] = -Sinang;
136     for (i1 = 0; i1 < 10; i1++)
137         for (j1 = 0; j1 < 10; j1++)
138             Rotation_result[i1][j1] = 0;
139     for (i1 = 0; i1 < r1; i1++)
140         for (j1 = 0; j1 < c2; j1++)
141             for (k1 = 0; k1 < c1; k1++)
142                 Rotation_result[i1][j1] = Rotation_result[i1][j1] + (p[i1][k1]
143 *Rotation_matrix[k1][j1]);
144     cout << "\n\n\t\tPolygon after Rotationâ€¦";
145     for (i = 0; i < n; i++)
146     {
147         cout << "\n";
148         for (int j = 0; j < 3; j++)
149         {
150             cout << Rotation_result[i][j] << "\t";
151         }
152         draw_polyfloat(Rotation_result, n);
153     }
154 void POLYGON ::scaling(int p[][10], int n)
155 {
156     float Sx, Sy;
157     int i, j;
158     int i1, j1, k1, r1, c1, c2;
159     r1 = n;
160     c1 = c2 = 3;
161     cout << "\n\n\t\tEnter X-Scaling Sx: ";
162     cin >> Sx;
163     cout << "\n\n\t\tEnter Y-Scaling Sy: ";
164     cin >> Sy;
165     for (i = 0; i < 3; i++)
166     {
167         for (j = 0; j < 3; j++)
168         {
169             Scaling_matrix[i][j] = 0;
170         }
171     }
172     Scaling_matrix[0][0] = Sx;
173     Scaling_matrix[0][1] = 0;
174     Scaling_matrix[0][2] = 0;
175     Scaling_matrix[1][0] = 0;

```

```

176     Scaling_matrix[1][1] = Sy;
177     Scaling_matrix[1][2] = 0;
178     Scaling_matrix[2][0] = 0;
179     Scaling_matrix[2][1] = 0;
180     Scaling_matrix[2][2] = 1;
181     for (i1 = 0; i1 < 10; i1++)
182         for (j1 = 0; j1 < 10; j1++)
183             Scaling_result[i1][j1] = 0;
184     for (i1 = 0; i1 < r1; i1++)
185         for (j1 = 0; j1 < c2; j1++)
186             for (k1 = 0; k1 < c1; k1++)
187                 Scaling_result[i1][j1] = Scaling_result[i1][j1] + (p[i1][k1]
188 *Scaling_matrix[k1][j1]);
189     cout << "\n\n\t\tPolygon after Scalingâ€¦";
190     draw_polyfloat(Scaling_result, n);
191 }
192 int main()
193 {
194     int ch, n, p[10][10];
195     POLYGON p1;
196     cout << "\n\n **** 2-D TRANSFORMATION ****";
197     n = p1.accept_poly(p);
198     cout << "\n\n\t\tOriginal Polygon â€¦";
199     p1.draw_poly(p, n);
200     do
201     {
202         int ch;
203         cout << "\n\n **** 2-D TRANSFORMATION ****";
204         cout << "\n\n\t\t1.Translation \n\n\t\t2.Scaling \n\n\t\t3.Rotation
205 \n\n\t\t4.Exit";
206         cout << "\n\n\tEnter your choice(1-6):";
207         cin >> ch;
208         switch (ch)
209         {
210             case 1:
211                 // cout<<"case1";
212                 p1.translation(p, n);
213                 break;
214             case 2:
215                 cout << "case2";
216                 p1.scaling(p, n);
217                 break;
218             case 3:
219                 cout << "case3";
220                 p1.rotation(p, n);
221                 break;
222             case 4:
223                 exit(0);
224         }
225     } while (1);
226     return 0;
227 }

```

```

/*Output:
**** 2-D TRANSFORMATION ****
Enter no.of vertices:3
Enter (x,y)Co-ordinate of point P0: 60
120
Enter (x,y)Co-ordinate of point P1: 120
192
Enter (x,y)Co-ordinate of point P2: 192
60
60 120 1
120 192 1
192 60 1
Original Polygon  $\Gamma\zeta^a$ 
**** 2-D TRANSFORMATION ****
1.Translation
2.Scaling
3.Rotation
4.Exit
Enter your choice(1-6):1
Enter X-Translation tx: 20
Enter Y-Translation ty: 30
Polygon after Translation  $\Gamma\zeta^a$ 
**** 2-D TRANSFORMATION ****
1.Translation
2.Scaling
3.Rotation
4.Exit
Enter your choice(1-6):2
case2
Enter X-Scaling Sx: 20
Enter Y-Scaling Sy: 30
Polygon after Scaling  $\Gamma\zeta^a$ 
**** 2-D TRANSFORMATION ****
1.Translation
2.Scaling
3.Rotation
4.Exit
Enter your choice(1-6):3
case3
Enter the angle of rotation in degrees: 60
**** Rotation Types ****
1.Clockwise Rotation
2.Anti-Clockwise Rotation
Enter your choice(1-2): 1
Mark1Mark2
Polygon after Rotation  $\Gamma\zeta^a$ 
133.923 8.03815 1
226.277 -7.9236 1
147.961 -136.277 1
**** 2-D TRANSFORMATION ****

```

1.Translation

2.Scaling

3.Rotation

4.Exit

Enter your choice(1-6):4 \*/