12/25/22, 3:36 PM A4.cpp

```
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:
       int p[10][10], Trans result[10][10], Trans matrix[10][10];
       float Rotation_result[10][10], Rotation_matrix[10][10];
11
       float Scaling_result[10][10], Scaling_matrix[10][10];
       float Shearing_result[10][10], Shearing_matrix[10][10];
12
       int Reflection_result[10][10], Reflection_matrix[10][10];
13
14
15 public:
16
       int accept_poly(int[][10]);
17
       void draw poly(int[][10], int);
       void draw_polyfloat(float[][10], int);
18
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:";</pre>
31
       cin >> n;
32
       for (i = 0; i < n; i++)
33
           cout << "\n\n\t\tEnter (x,y)Co-ordinate of point P" << i << ": ";</pre>
34
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";
           for (int j = 0; j < 3; j++)
41
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);
```

localhost:4649/?mode=clike

```
12/25/22 3:36 PM
                                                      A4.cpp
  59
             }
  60
             else
                  line(p[i][0] + 320, -p[i][1] + 240, p[0][0] + 320, -p[0][1] + 240);
  61
  62
         delay(3000);
  63
  64 }
  65 void POLYGON ::draw polyfloat(float p[][10], int n)
  66 {
         int i, gd = DETECT, gm;
  67
  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;
         cout << "\n\n\t\tEnter X-Translation tx: ";</pre>
  88
  89
         cin >> tx;
  90
         cout << "\n\n\t\tEnter Y-Translation ty: ";</pre>
  91
         cin >> ty;
  92
         for (i = 0; i < 3; i++)
             for (j = 0; j < 3; j++)
  93
  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++)
             for (j1 = 0; j1 < c2; j1++)
 102
                  for (k1 = 0; k1 < c1; k1++)
 103
                      Trans_result[i1][j1] = Trans_result[i1][j1] + (p[i1][k1] *
 104
     Trans_matrix[k1][j1]);
         cout << "\n\n\t\tPolygon after Translation…";</pre>
 105
 106
         draw poly(Trans result, n);
 107 }
 108 void POLYGON ::rotation(int p[][10], int n)
 109 {
 110
         float type, Ang, Sinang, Cosang;
         int i, j;
 111
         int i1, j1, k1, r1, c1, c2;
 112
 113
         r1 = n;
 114
         c1 = c2 = 3;
 115
         cout << "\n\n\t\tEnter the angle of rotation in degrees: ";</pre>
 116
         cin >> Ang;
```

localhost:4649/?mode=clike 2/4

12/25/22, 3:36 PM A4.cpp cout << "\n\n \*\*\*\* Rotation Types \*\*\*\*";</pre> 117 cout << "\n\n\t\t1.Clockwise Rotation \n\n\t\t2.Anti-Clockwise Rotation ";</pre> 118 119 cout << "\n\n\t\tEnter your choice(1-2): ";</pre> 120 cin >> type; Ang = (Ang \* 6.2832) / 360;121 122 Sinang = sin(Ang); 123 Cosang = cos(Ang); 124 cout << "Mark1";</pre> for (i = 0; i < 3; i++)125 126 for (j = 0; j < 3; j++)127 Rotation\_matrix[i][j] = 0; cout << "Mark2";</pre> 128 129 Rotation matrix[0][0] = Rotation matrix[1][1] = Cosang; Rotation\_matrix[0][1] = Rotation\_matrix[1][0] = Sinang; 130 131 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++)for (j1 = 0; j1 < 10; j1++)137 138 Rotation\_result[i1][j1] = 0; 139 for (i1 = 0; i1 < r1; i1++)for (j1 = 0; j1 < c2; j1++)140 141 for (k1 = 0; k1 < c1; k1++)Rotation\_result[i1][j1] = Rotation\_result[i1][j1] + (p[i1][k1] 142 \*Rotation\_matrix[k1][j1]); 143 cout << "\n\n\t\tPolygon after Rotation…"; 144 for (i = 0; i < n; i++)145 cout << "\n"; 146 for (int j = 0; j < 3; j++) 147 148 { cout << Rotation\_result[i][j] << "\t";</pre> 149 150 } 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; int i1, j1, k1, r1, c1, c2; 158 159 r1 = n;160 c1 = c2 = 3;cout << "\n\n\t\tEnter X-Scaling Sx: ";</pre> 161 162 cin >> Sx; 163 cout << "\n\n\t\tEnter Y-Scaling Sy: ";</pre> 164 cin >> Sy; 165 for (i = 0; i < 3; i++)166 for (j = 0; j < 3; j++)167 168 { 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;

localhost:4649/?mode=clike 3/4

12/25/22, 3:36 PM A4.cpp

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

localhost:4649/?mode=clike 4/4

```
/*Output:
**** 2-D TRANSFORMATION ****
Enter no. of vertices:3
Enter (x,y)Co-ordinate of point P0: 60
Enter (x,y)Co-ordinate of point P1: 120
Enter (x,y)Co-ordinate of point P2: 192
60
60 120 1
120 192 1
192 60 1
Original Polygon ΓÇ<sup>a</sup>
**** 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 TranslationΓÇ<sup>a</sup>
**** 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 ScalingΓÇ<sup>a</sup>
**** 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 RotationΓÇ<sup>a</sup>
133.923 8.03815 1
226.277 -7.9236 1
147.961 -136.277 1
**** 2-D TRANSFORMATION ****
```

- 1.Translation2.Scaling3.Rotation

- 4.Exit

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