/*

Title: - Write C++ program to draw 2-D object and perform following basic transformations, Scaling b) Translation c) Rotation. Apply the concept of operator overloading.

Roll No:-

Class:-SE Computer

Sub:-OOPL & CGL

Date:-

**********/

Program-

```
#include<iostream>
#include<stdlib.h>
#include<graphics.h>
#include<math.h>

using namespace std;

class POLYGON
{
   private:
      int p[10][10],Trans_result[10][10],Trans_matrix[10][10];
      float Rotation_result[10][10],Rotation_matrix[10][10];
      float Scaling_result[10][10],Scaling_matrix[10][10];
      float Shearing_result[10][10],Shearing_matrix[10][10];
      int Reflection_result[10][10],Reflection_matrix[10][10];

   public:
         int accept_poly(int [][10]);
         void draw_poly(int [][10],int);
         void draw_polyfloat(float [][10],int);
         void matmult(int [][10],int [][10],int,int,int,int [][10]);
         void matmultfloat(float [][10],int [][10],int,int,int,float [][10]);
         void shearing(int [][10],int);
         void scaling(int [][10],int);
         void rotation(int [][10],int);
         void translation(int [][10],int);
         void reflection(int [][10],int);
};

int POLYGON :: accept_poly(int p[][10])
{
         int i,n;
         cout<<"\n\n\t\tEnter no.of vertices:";
```

```
	cin>>n;
	for(i=0;i<n;i++)
	{
		cout<<"\n\n\t\tEnter (x,y)Co-ordinate of point P"<<i<<": ";
		cin >> p[i][0] >> p[i][1];
		p[i][2] = 1;
	}

	for(i=0;i<n;i++)
	{
		cout<<"\n";
		for(int j=0;j<3;j++)
		{
			cout<<p[i][j]<<"\t";
		}
	}

	return n;
}
void POLYGON :: draw_poly(int p[][10], int n)
{
	int i,gd = DETECT,gm;
	initgraph(&gd,&gm,NULL);
	line(320,0,320,480);
	line(0,240,640,240);

	for(i=0;i<n;i++)
	{
		if(i<n-1)
		{
			line(p[i][0]+320, -p[i][1]+240, p[i+1][0]+320, -p[i+1][1]+240);

		}
		else
			line(p[i][0]+320, -p[i][1]+240, p[0][0]+320, -p[0][1]+240);
	}
	delay(3000);


}

void POLYGON :: draw_polyfloat(float p[][10], int n)
{
	int i,gd = DETECT,gm;
	initgraph(&gd,&gm,NULL);
	line(320,0,320,480);
```

```
	line(0,240,640,240);

	for(i=0;i<n;i++)
	{
		if(i<n-1)
		{
			line(p[i][0]+320, -p[i][1]+240, p[i+1][0]+320, -p[i+1][1]+240);

		}
		else
			line(p[i][0]+320, -p[i][1]+240, p[0][0]+320, -p[0][1]+240);
	}
	//delay(8000);

}


void POLYGON :: translation(int p[10][10],int n)
{
	int tx,ty,i,j; int i1,j1,k1,r1,c1,c2;
      r1=n;c1=c2=3;
	cout << "\n\n\t\tEnter X-Translation tx: ";
	cin >> tx;
	cout << "\n\n\t\tEnter Y-Translation ty: ";
	cin >> ty;
	for(i=0;i<3;i++)
	for(j=0;j<3;j++)
		Trans_matrix[i][j] = 0;
	Trans_matrix[0][0] = Trans_matrix[1][1] = Trans_matrix[2][2] = 1;
	Trans_matrix[2][0] = tx;
	Trans_matrix[2][1] = ty;

	for(i1=0;i1<10;i1++)
	for(j1=0;j1<10;j1++)
		Trans_result[i1][j1] = 0;
	for(i1=0;i1<r1;i1++)
	for(j1=0;j1<c2;j1++)
	for(k1=0;k1<c1;k1++)
		Trans_result[i1][j1] = Trans_result[i1][j1]+(p[i1][k1] * Trans_matrix[k1][j1]);
	cout << "\n\n\t\tPolygon after Translationâ€¦";
	draw_poly(Trans_result,n);
}

void POLYGON :: rotation(int p[][10],int n)
{
	float type,Ang,Sinang,Cosang;
```

```
	int i,j; int i1,j1,k1,r1,c1,c2;
	r1=n;c1=c2=3;
	   cout << "\n\n\t\tEnter the angle of rotation in degrees: ";
	   cin >> Ang;
	   cout << "\n\n **** Rotation Types ****";
	   cout << "\n\n\t\t1.Clockwise Rotation \n\n\t\t2.Anti-Clockwise Rotation ";
	   cout << "\n\n\t\tEnter your choice(1-2): ";
	   cin >> type;
	   Ang = (Ang * 6.2832)/360;
	   Sinang = sin(Ang);
	   Cosang = cos(Ang);
	  cout<<"Mark1";
	   for(i=0;i<3;i++)
	   for(j=0;j<3;j++)
			Rotation_matrix[i][j] = 0;
	cout<<"Mark2";
	   Rotation_matrix[0][0] = Rotation_matrix[1][1] = Cosang;
	   Rotation_matrix[0][1] = Rotation_matrix[1][0] = Sinang;
	   Rotation_matrix[2][2] = 1;
	   if(type == 1)
			Rotation_matrix[0][1] = -Sinang;
	   else
			Rotation_matrix[1][0] = -Sinang;

	for(i1=0;i1<10;i1++)
	   for(j1=0;j1<10;j1++)
			Rotation_result[i1][j1] = 0;
	   for(i1=0;i1<r1;i1++)
	   for(j1=0;j1<c2;j1++)
	   for(k1=0;k1<c1;k1++)
			Rotation_result[i1][j1] = Rotation_result[i1][j1]+(p[i1][k1] *
Rotation_matrix[k1][j1]);

	   cout << "\n\n\t\tPolygon after Rotationâ€¦";
	for(i=0;i<n;i++)
	   {
			cout<<"\n";
			for(int j=0;j<3;j++)
			{
				cout<<Rotation_result[i][j]<<"\t";
			}
	   }
	   draw_polyfloat(Rotation_result,n);
}

void POLYGON :: scaling(int p[][10],int n)
```

```
{
        float Sx,Sy;
      int i,j; int i1,j1,k1,r1,c1,c2;
      r1=n;c1=c2=3;
        cout<<"\n\n\t\tEnter X-Scaling Sx: ";
        cin>>Sx;
        cout<<"\n\n\t\tEnter Y-Scaling Sy: ";
        cin>>Sy;

        for(i=0;i<3;i++)
        {
                for(j=0;j<3;j++)
                {
                        Scaling_matrix[i][j] = 0;
                }
        }

        Scaling_matrix[0][0] = Sx;
        Scaling_matrix[0][1] = 0;
        Scaling_matrix[0][2] = 0;
        Scaling_matrix[1][0] = 0;
        Scaling_matrix[1][1] = Sy;
        Scaling_matrix[1][2] = 0;
        Scaling_matrix[2][0] = 0;
        Scaling_matrix[2][1] = 0;
        Scaling_matrix[2][2] = 1;

      for(i1=0;i1<10;i1++)
        for(j1=0;j1<10;j1++)
                Scaling_result[i1][j1] = 0;
        for(i1=0;i1<r1;i1++)
        for(j1=0;j1<c2;j1++)
        for(k1=0;k1<c1;k1++)
                Scaling_result[i1][j1] = Scaling_result[i1][j1]+(p[i1][k1] *
Scaling_matrix[k1][j1]);

        cout<<"\n\n\t\tPolygon after Scalingâ€¦";
        draw_polyfloat(Scaling_result,n);
}


int main()
{
        int ch,n,p[10][10];
        POLYGON p1;
        cout<<"\n\n **** 2-D TRANSFORMATION ****";
```

```
        n= p1.accept_poly(p);

        cout <<"\n\n\t\tOriginal Polygon â€¦";
        p1.draw_poly(p,n);
        do
        {

           int ch;
              cout<<"\n\n **** 2-D TRANSFORMATION ****";
                 cout<<"\n\n\t\t1.Translation \n\n\t\t2.Scaling \n\n\t\t3.Rotation \n\n\t\t4.Exit";
              cout<<"\n\n\tEnter your choice(1-6):";
              cin>>ch;

                 switch(ch)
                 {
                          case 1:
                                   //cout<<"case1";
                                   p1.translation(p,n);
                                   break;

                          case 2:
                                   cout<<"case2";
                                   p1.scaling(p,n);
                                   break;

                          case 3:
                                   cout<<"case3";
                                   p1.rotation(p,n);
                                   break;

                          case 4:
                                   exit(0);
                 }
        }while(1);
        return 0;
}
/*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 ΓÇª

**** 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 TranslationΓÇª

**** 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 ScalingΓÇª

**** 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 RotationΓÇª
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 */