Assignment No: 03

Title/ Problem Statement:

Write C++/Java program to draw 2-D object and perform following basic transformations,
a) Scaling
b) Translation
c) Rotation
Use operator overloading

Prerequisites: Matrix operations, such as addition, subtraction, multiplication

Objectives: To study basic object transformations using matrix operations.

Theory :

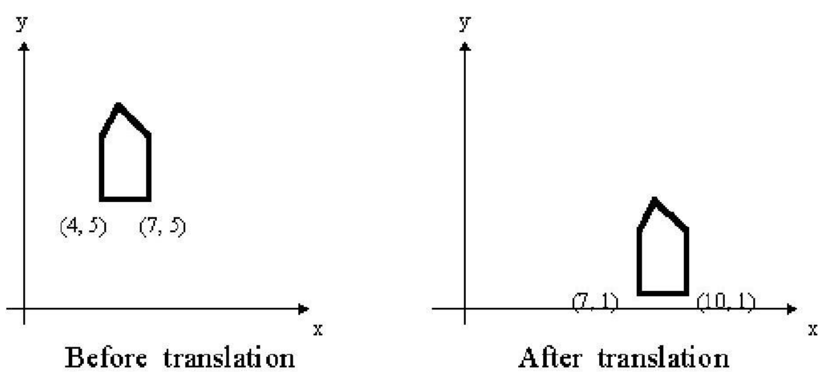
Aim: To apply the basic 2D transformations such as translation, Scaling, Rotation, shearing and reflection for a given 2D object.

Description: We have to perform 2D transformations on 2D objects. Here we perform transformations on a line segment.

The 2D transformations are:

1. Translation
2. Scaling
3. Rotation
4. Reflection
5. Shear

1. Translation: Translation is defined as moving the object from one position to another position along straight line path.



We can move the objects based on translation distances along x and y axis. tx denotes translation distance along x-axis and ty denotes translation distance along y axis.

Translation Distance: It is nothing but by how much units we should shift the object from one location to another along x, y-axis.

Consider (x,y) are old coordinates of a point. Then the new coordinates of that same point (x’,y’) can be obtained as follows:

X’=x+tx

Y’=y+ty

We denote translation transformation as P. we express above equations in matrix form as:

$$P' = P + T$$

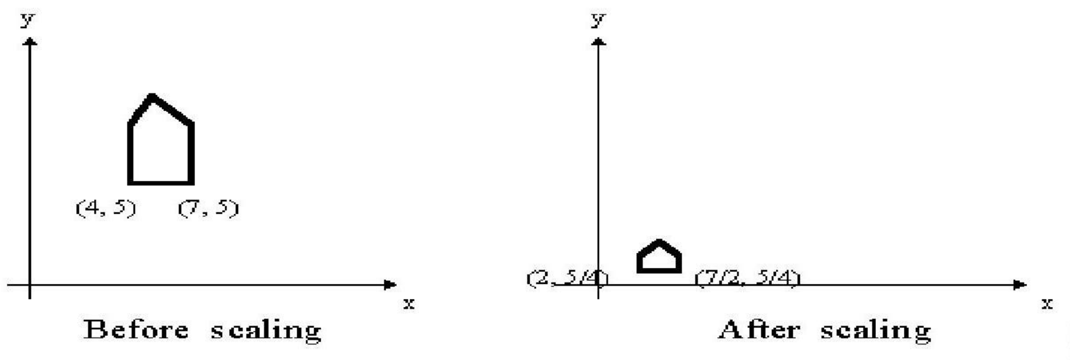
$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

x,y---old coordinates

x’,y’—new coordinates after translation

tx,ty—translation distances, T is

2. Scaling: scaling refers to changing the size of the object either by increasing or decreasing. We will increase or decrease the size of the object based on scaling factors along x and y-axis.



If (x, y) are old coordinates of object, then new coordinates of object after applying scaling transformation are obtained as:

x’=x*sx

y’=y*sy.

sx and sy are scaling factors along x-axis and y-axis. we express the above equations in matrix form as:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Scaling Matrix

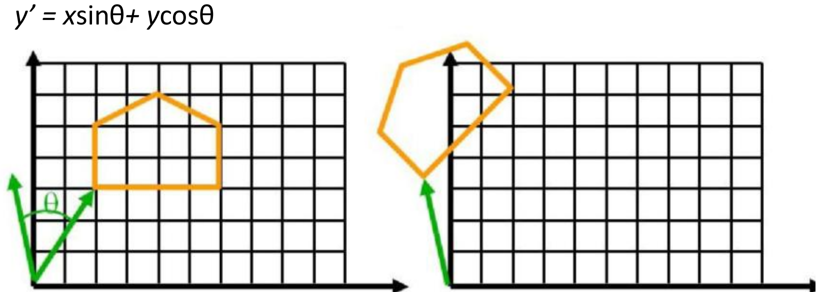
3. **Rotation:** A rotation repositions all points in an object along a circular path in the plane centered at the pivot point. We rotate an object by an angle theta.

New coordinates after rotation depend on *both x* and *y*

$x' = x\cos\theta - y\sin\theta$

$y' = x\sin\theta + y\cos\theta$

or in matrix form:



P' = R • P,

$$R = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

R-rotation matrix.

Algorithm:

```
#include<iostream>
#include<stdlib.h>
#include<graphics.h> //header file for switching text to graphics mode
#include<math.h>

using namespace std;
class transform
{
  private:
    int edges,sx,sy,tx,ty;
    int input[20][2],scalm[2][2];
    float rot[2][2],theta,resmr[20][2];
    int resm[20][2];
    int trans[2][2],clockw;

 public:
    void menu();
    void accept();
    void scale();
    void translate();
    void rotate();
    void multiply(int a[20][2],int b[2][2],int c[20][2]);
    void multiply1(int a[20][2],float b[2][2],float c[20][2]);
    void plot(int mat[20][2]);
    void plot1(float mat[20][2]);

};

void transform::accept()
{

  int i,j;
  cout<<"\n Enter no. of edges in figure:";
```

```
  cin>>edges;
  cout<<"Enter co-ordinates in matrix form:";
  for(i=1;i<=edges;i++)
   {
     for(j=1;j<=2;j++)
      {
        cout<<"\nA["<<i<<"]["<<j<<"]=";
        cin>>input[i][j];
      }
   }
   plot(input);
}

void transform::plot(int mat[20][2])
{
   int i;
   line(0,240,640,240);
   line(320,0,320,480);
   for(i=1;i < edges;i++)
         line(320+mat[i][1],240-mat[i][2],320+mat[i+1][1],240-mat[i+1][2]);
   line(320+mat[1][1],240-mat[1][2],320+mat[i][1],240-mat[i][2]);
}

void transform::plot1(float mat[20][2])
{
   int i;
   line(0,240,640,240);
   line(320,0,320,480);
   for(i=1;i < edges;i++)
         line(320+mat[i][1],240-mat[i][2],320+mat[i+1][1],240-mat[i+1][2]);
   line(320+mat[1][1],240-mat[1][2],320+mat[i][1],240-mat[i][2]);
}


int main(void)
{
  char ch;
  int gd=DETECT,gm,i;
  initgraph(&gd,&gm,NULL);
  do
    {

       transform t;
       t.menu();
       cout<<"\n Do you want to continue::";
       cin>>ch;
    }while(ch=='y'||ch=='Y');
   return 0;
}


void transform::menu()
{
```

```
  int ch;
  cout<<"______________________";
  cout<<"\n1. Scaling::";
  cout<<"\n2. Translation::";
  cout<<"\n3. Rotation::\n";
  cout<<"______________________";
  cout<<"\nEnter ur choice::\t";
  cin>>ch;

   switch(ch)
   {
    case 1:
        scale();
        break;

    case 2:
        translate();
        break;

    case 3:
     rotate();
     break;

    default: exit(0);
   }
}

void transform::rotate()
{
 int i;
 float theta1;
 cleardevice();
 accept();   //accept the original polygon
 cout<<"\n Enter the angle for rotation:";
 cin>>theta;   //accept angle for rotation
 cout<<"\n Enter 1 for clockwise rotation 0 for anti clockwise rotation:";
 cin>>clockw;
 theta1=((3.14*theta)/180);  //conversion of degree to radian
 if(clockw==1)
  {
   rot[1][1]=rot[2][2]=cos(theta1);
   rot[1][2]=-sin(theta1);
   rot[2][1]=sin(theta1);
  }
 else
  {
   rot[1][1]=rot[2][2]=cos(theta1);
   rot[1][2]=sin(theta1);
   rot[2][1]=-sin(theta1);
  }
 multiply1(input,rot,resmr);
 plot1(resmr);
}
```

```
void transform::scale()
{
  int i;
  cleardevice();
  accept();
  cout<<"\n Enter the scale X factor:";
  cin>>sx;
  cout<<"\n Enter the scale Y factor:";
  cin>>sy;
  scalm[1][1]=sx;
  scalm[1][2]=scalm[2][1]=0;
  scalm[2][2]=sy;
  multiply(input,scalm,resm);
  plot(resm);
 }

void transform::translate()
{
  int i,j;
  cleardevice();
  accept();
  cout<<"/n Enter tx factor";
  cin>>tx;
  cout<<"/n Enter ty factor";
  cin>>ty;
  for(i=1;i<=edges;i++)
  {
     input[i][1]=  input[i][1]+tx;
     input[i][2]=  input[i][2]+ty;
  }
   plot(input);
}

void transform::multiply(int a[20][2],int b[2][2],int c[20][2])
{
  int i;
  for(i=1;i<=edges;i++)
   {
     c[i][1]=(a[i][1]*b[1][1])+(a[i][2]*b[2][1]);
     c[i][2]=(a[i][1]*b[1][2])+(a[i][2]*b[2][2]);
   }
}


void transform::multiply1(int a[20][2],float b[2][2],float c[20][2])
{
  int i;
  for(i=1;i<=edges;i++)
   {
     c[i][1]=(a[i][1]*b[1][1])+(a[i][2]*b[2][1]);
     c[i][2]=(a[i][1]*b[1][2])+(a[i][2]*b[2][2]);
```

```
  }
}
```

Input/Output-

```
isbm@isbm-ThinkCentre-M72e:~/cg$ g++ ass3.cpp -lgraph
isbm@isbm-ThinkCentre-M72e:~/cg$ ./a.out
______________________
1. Scaling::
2. Translation::
3. Rotation::
______________________
Enter ur choice:: 2

 Enter no. of edges in figure:3
Enter co-ordinates in matrix form:
A[1][1]=20

A[1][2]=50

A[2][1]=70

A[2][2]=100

A[3][1]=100

A[3][2]=20
/n Enter tx factor3
/n Enter ty factor3

 Do you want to continue::y
______________________
1. Scaling::
2. Translation::
3. Rotation::
______________________
Enter ur choice:: 1

 Enter no. of edges in figure:3
Enter co-ordinates in matrix form:
A[1][1]=20

A[1][2]=50

A[2][1]=70

A[2][2]=100

A[3][1]=100

A[3][2]=20
```

Enter the scale X factor:2

Enter the scale Y factor:2

Do you want to continue::y

____

1. Scaling::
2. Translation::
3. Rotation::

____

Enter ur choice:: 3

Enter no. of edges in figure:3

Enter co-ordinates in matrix form:

A[1][1]=20

A[1][2]=50

A[2][1]=70

A[2][2]=100

A[3][1]=100

A[3][2]=20

Enter the angle for rotation:45

Enter 1 for clockwise rotation 0 for anti clockwise rotation:1

Do you want to continue::n

Conclusion: Thus we have implemented program to perform scalling, translation and rotation operation.