# Experiment- 10

## Aim

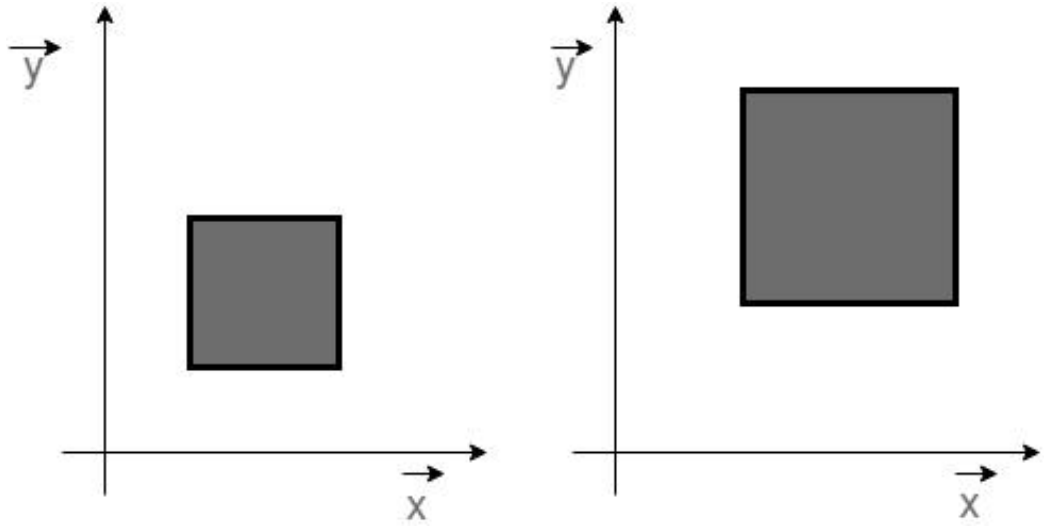
Program to implement Scaling of an object

## Description/Theory

A scaling transformation alters size of an object. In the scaling process, we either compress or expand the dimension of the object.
Scaling operation can be achieved by multiplying each vertex coordinate (x, y) of the polygon by scaling factor $s_x$and $s_y$ to produce the transformed coordinates as (x', y').
So, x' = x * $s_x$ and y' = y * $s_y$.
The scaling factor $s_x$, $s_y$ scales the object in X and Y direction respectively. So, the above equation can be represented in matrix form:

$$\begin{bmatrix} X' \\ Y' \end{bmatrix} = \begin{bmatrix} Sx & 0 \\ 0 & Sy \end{bmatrix} \begin{bmatrix} X \\ Y \end{bmatrix}$$

Or P' = S . P
Scaling process:




## Algorithm

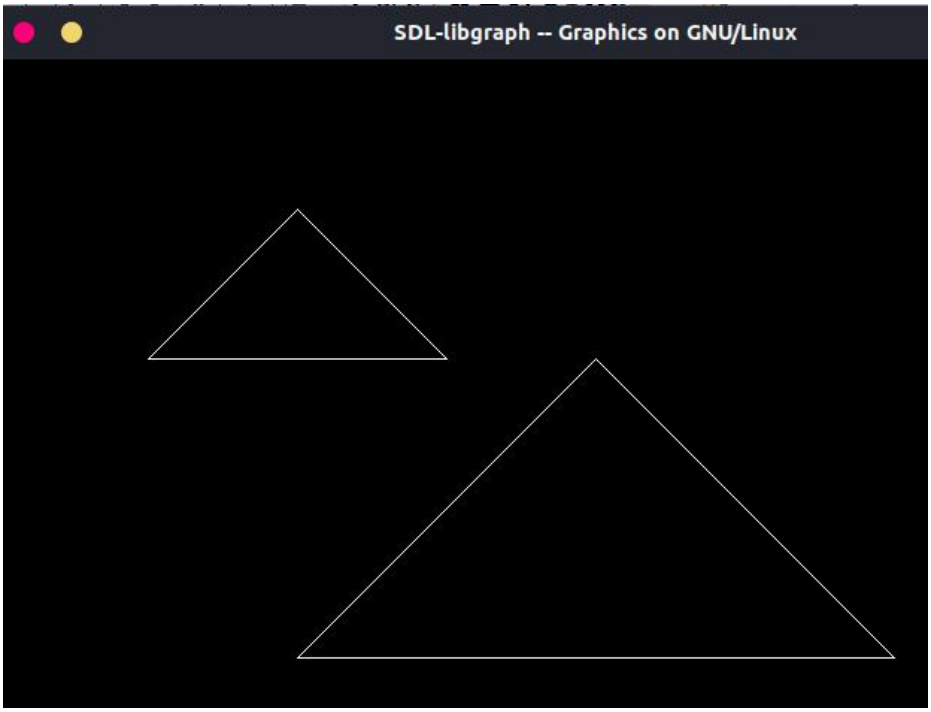
The algorithm for Scaling of an object is given below:

1) Make a 2x2 scaling matrix S as:
   $S_x$ 0
   0 $S_y$
2) For each point of the polygon.
   a) Make a 2x1 matrix P, where P[0][0] equals to x coordinate of the point and P[1][0] equals to y coordinate of the point.
   b) Multiply scaling matrix S with point matrix P to get the new coordinate.
3) Draw the polygon using new coordinates.

## Code

```
#include<stdio.h>
#include<graphics.h>

void findNewCoordinate(int s[][2], int p[][1])
{
   int temp[2][1] = { 0 };

   for (int i = 0; i < 2; i++)
      for (int j = 0; j < 1; j++)
         for (int k = 0; k < 2; k++)
            temp[i][j] += (s[i][k] * p[k][j]);

   p[0][0] = temp[0][0];
   p[1][0] = temp[1][0];
}
void scale(int x[], int y[], int sx, int sy)
{
   line(x[0], y[0], x[1], y[1]);
   line(x[1], y[1], x[2], y[2]);
   line(x[2], y[2], x[0], y[0]);

   int s[2][2] = { sx, 0, 0, sy };
   int p[2][1];

   for (int i = 0; i < 3; i++)
   {
      p[0][0] = x[i];
      p[1][0] = y[i];
      findNewCoordinate(s, p);
      x[i] = p[0][0];
      y[i] = p[1][0];
   }
   line(x[0], y[0], x[1], y[1]);
   line(x[1], y[1], x[2], y[2]);
   line(x[2], y[2], x[0], y[0]);
}

int main()
{
   int x[] = { 100, 200, 300 };
   int y[] = { 200, 100, 200 };
   int sx = 2, sy = 2;
   int gd, gm;
   detectgraph(&gd, &gm);
   initgraph(&gd, &gm," ");
   scale(x, y, sx,sy);
   getch();
   return 0;
}
```

## Result




## Discussion

The object, a triangle in this case, on the left side is the original object. And the enlarged image on the right side is the same triangle, scaled up to larger co-ordinate values. Representing in form of a matrix, the co-ordinates are individually multiplied by the Scaling Factors to “enlarge” or “reduce” the object.

## Finding and Learning

A scaling transform can be used to make objects bigger or smaller. Mathematically, a scaling transform simply multiplies each x-coordinate by a given amount and each y-coordinate by a given amount. That is, if a point (*x*,*y*) is scaled by a factor of *a* in the x direction and by a factor of *d* in the y direction, then the resulting point (*x1*,*y1*) is given by

x1 = a * x
y1 = d * y

If you apply this transform to a shape that is centred at the origin, it will stretch the shape by a factor of *a* horizontally and *d* vertically.
When scaling is applied to a shape that is not centred at (0,0), then in addition to being stretched or shrunk, the shape will be moved away from 0 or towards 0. In fact, the true description of a scaling operation is that it pushes every point away from (0,0) or pulls every point towards (0,0). If you want to scale about a point other than (0,0), you can use a sequence of three transforms, similar to what was done in the case of rotation.

# Experiment- 11

## Aim

Program to implement Reflection of an object.

## Description/Theory

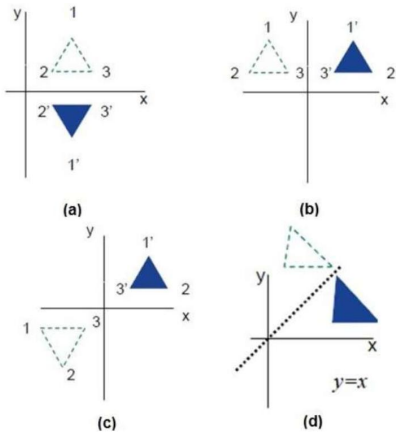
Reflection is nothing more than a rotation of the object by 180º. In case of reflection the image formed is on the opposite side of the reflective medium with the same size. Therefore we use the identity matrix with positive and negative signs according to the situation respectively.

The reflection about the *x-axis* can be shown as:

$$T = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$$

The reflection about the *y-axis* can be shown as:

$$T = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$$




## Algorithm

The algorithm for reflection about any arbitrary line $y = mx + c$ can be accomplished with a combination of translate-rotate-reflect transformations.

Steps are as follows:

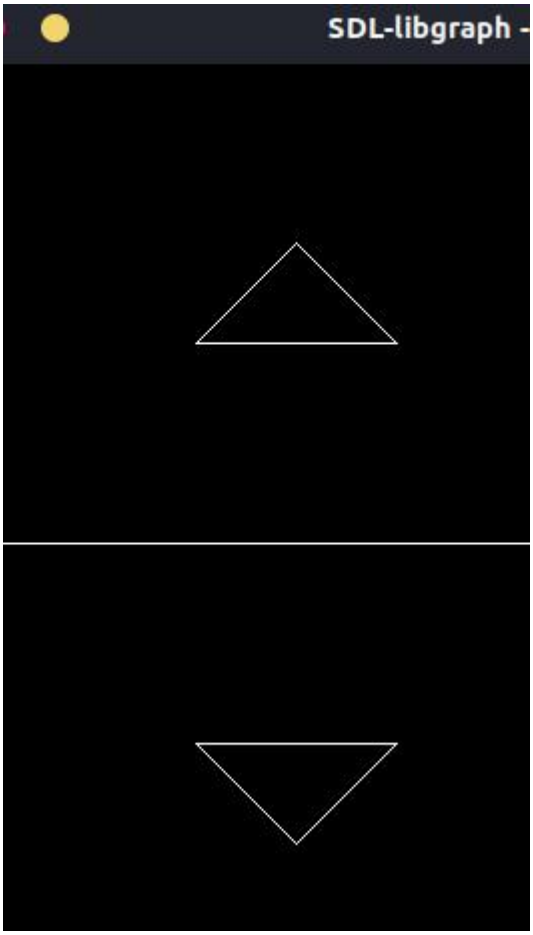
1. Translate the working coordinate system (WCS) so that the line passes through the origin.
2. Rotate the WCS such that one of the coordinate axis lies onto the line.
3. Reflect about the aligned axis
4. Restore the WCS back by using the inverse rotation and translation transformation.

## Code

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

int main()
{
	int gd = DETECT,gm;
	initgraph(&gd, &gm, "");

	int a = getmaxx();
	int b = getmaxy();

	int y = b/2;
	int x = a/2;

	line(0, y, 300, y);
	int x1 = x + 100;
	int y1 = y - 100;
```

```
        int x2 = x + 150;
        int y2 = y - 150;

        line(x - 100, y + 100, x - 200, y + 100);
        line(x - 150, y + 150, x - 200, y + 100);
        line(x - 100, y + 100, x – 150, y + 150);
        line(x - 100, y - 100, x - 150, y - 150);
        line(x - 150, y - 150, x - 200, y - 100);
        line(x - 200, y - 100, x - 100, y - 100);

        getch();
        closegraph();
        return 0;
}
```

## Result




## Discussion

In the output as shown, the horizontal line acts as the mirror about which the object, a triangle in this case, is reflected i.e. translated and then rotated by 180 degrees.

## Finding and Learning

Reflection is the mirror image of original object. In other words, we can say that it is a rotation operation with 180°. In reflection transformation, the size of the object does not change.