

()



Ex. No: 1

Date : _____

Implementation of Algorithms for drawing 2D Primitives

Aim:

To implement the algorithms for drawing a line using DDA algorithm and Bresenham algorithm, and also drawing a circle using midpoint algorithm.

Algorithm:

1. Print the menu for choosing 2D primitive.
2. If user choose line then print the menu for choosing algorithm and get the start point (xa, ya), end point (xb, yb) of line.
 - a. If user choose DDA algorithm then call the DDA line drawing algorithm.
 - b. If user choose Bresenham algorithm then call the Bresenham line drawing algorithm.
3. If user choose circle then get the center point (xCenter, yCenter), radius of the circle and call the midpoint circle drawing algorithm.

DDA Line Drawing Algorithm:

1. Calculate $dx = x_b - x_a$, $dy = y_b - y_a$ and assign $x = x_a$, $y = y_a$.
2. a) If $\text{abs}(dx) > \text{abs}(dy)$ then set steps in $\text{abs}(dx)$ times.
b) Otherwise set steps in $\text{abs}(dy)$ times.
3. Calculate the $x\text{Increment} = dx / \text{steps}$ and $y\text{Increment} = dy / \text{steps}$.
4. Plot the coordinate values (x, y).
5. Initially $k = 0$.
6. Plot the coordinate values $(x + x\text{Increment}, y + y\text{Increment})$ until $k \geq \text{steps}$.

Bresenham's Line Drawing Algorithm:

1. Calculate $dx = \text{abs}(x_a - x_b)$, $dy = \text{abs}(y_a - y_b)$, $p = 2 * dy - dx$, $\text{twoDy} = 2 * dy$, $\text{twoDyDx} = 2 * (dy - dx)$.
2. a) If $x_a > x_b$ then set $x = x_b$, $y = y_b$ and $x\text{End} = x_a$.
b) Otherwise set $x = x_a$, $y = y_a$, $x\text{End} = x_b$.
3. Plot the coordinate values (x, y).
4. a) If $p < 0$, then plot the coordinate values (x+1, y) and set $p = p + \text{twoDy}$.
b) Otherwise plot the coordinate values (x+1, y+1) and set $p = p + \text{twoDyDx}$.
5. Repeat the step 4 until $x \geq x\text{End}$.

Midpoint Circle Drawing Algorithm:

1. Assign $x = 0$, $y = \text{radius}$ and $p = 1 - \text{radius}$.
2. Plot the circle plot points in (xCenter, yCenter, x, y).
3. a) If $p < 0$ then plot the circle plot points in (xCenter, yCenter, x+1, y) and set $p = p + 2 * x + 1$.
b) Otherwise plot the circle plot points in (xCenter, yCenter, x+1, y-1) and set $p = p + 2 * (x - y) + 1$.
4. Repeat the step 3 until $x \geq y$.

Program:

```

#include <graphics.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>
#define ROUND(a) ((int)(a+0.5))
int xa, ya, xb, yb, xCenter, yCenter, radius, gd, gm, ch, ch2;
void lineDDA (int, int, int, int);
void lineBres (int, int, int, int);
void circleMidpoint (int, int, int);
void circlePlotPoints (int, int, int, int);
void main ()
{
    clrscr ();
    do
    {
        cout << "\n\t Implementation of Algorithms for drawing 2D Primitives";
        cout << "\n\n Choose any one 2D Primitive: ";
        cout << "\n\t 1.Line \n\t 2.Circle \n Enter your choice: \t";
        cin >> ch;
        switch (ch)
        {
            case 1:
                cout << "\n\n Choose any one line drawing algorithm: ";
                cout << "\n\t 1.DDA Algorithm \n\t 2.Bresenham Algorithm";
                cout << "\n Enter your choice: \t";
                cin >> ch2;
                cout << "\n Enter the start point of line (xa,ya): \t";
                cin >> xa >> ya;
                cout << "\n Enter the end point of line (xb,yb): \t\t";
                cin >> xb >> yb;
                switch (ch2)
                {
                    case 1:
                        initgraph (&gd, &gm, "C:\\TC\\BGI");
                        lineDDA (xa, ya, xb, yb);
                        closegraph ();
                        break;
                    case 2:
                        initgraph (&gd, &gm, "C:\\TC\\BGI");
                        lineBres (xa, ya, xb, yb);
                        closegraph ();
                        break;
                }
            break;
            case 2:

```

```

        cout << "\n Enter the center point of the circle: \t";
        cin >> xCenter >> yCenter;
        cout << "\n Enter the radius of the circle: \t";
        cin >> radius;
        initgraph (&gd, &gm, "C:\\TC\\BGI");
        circleMidpoint (xCenter, yCenter, radius);
        closegraph ();
        break;
    }
}
while (ch < 3);
getch ();
}
void lineDDA (int xa, int ya, int xb, int yb)
{
    int dx = xb - xa, dy = yb - ya, steps, k;
    float xIncrement, yIncrement, x = xa, y = ya;
    if(abs (dx) > abs (dy))
        steps = abs (dx);
    else
        steps = abs (dy);
    xIncrement = dx / (float) steps;
    yIncrement = dy / (float) steps;
    putpixel (ROUND(x), ROUND(y), 15);
    for (k=0; k<steps; k++)
    {
        x += xIncrement;
        y += yIncrement;
        putpixel (ROUND(x), ROUND(y), 15);
    }
    getch();
}
void lineBres (int xa, int ya, int xb, int yb)
{
    int dx = abs (xa - xb), dy = abs (ya - yb), p = 2 * dy - dx, twoDy = 2 * dy, twoDyDx = 2 *
(dy - dx), x, y, xEnd;
    if (xa > xb)
    {
        x = xb;
        y = yb;
        xEnd = xa;
    }
    else
    {
        x = xa;
        y = ya;

```

```

        xEnd = xb;
    }
    putpixel (x, y, 15);
    while (x < xEnd)
    {
        x++;
        if(p < 0)
            p += twoDy;
        else
        {
            y++;
            p += twoDyDx;
        }
        putpixel (x, y, 15);
    }
    getch();
}

void circleMidpoint(int xCenter, int yCenter, int radius)
{
    int x = 0, y = radius, p = 1 - radius;
    circlePlotPoints (xCenter, yCenter, x, y);
    while (x < y)
    {
        x++;
        if(p < 0)
            p += 2 * x + 1;
        else
        {
            y--;
            p += 2 * (x - y) + 1;
        }
        circlePlotPoints (xCenter, yCenter, x, y);
    }
    getch();
}

void circlePlotPoints (int xCenter, int yCenter, int x, int y)
{
    putpixel (xCenter + x, yCenter + y, 15);
    putpixel (xCenter - x, yCenter + y, 15);
    putpixel (xCenter + x, yCenter - y, 15);
    putpixel (xCenter - x, yCenter - y, 15);
    putpixel (xCenter + y, yCenter + x, 15);
    putpixel (xCenter - y, yCenter + x, 15);
    putpixel (xCenter + y, yCenter - x, 15);
    putpixel (xCenter - y, yCenter - x, 15);
}

```

Output:

Case 1:

Implementation of Algorithms for drawing 2D Primitives

Choose any one 2D Primitive:

1. Line
2. Circle

Enter your choice: 1

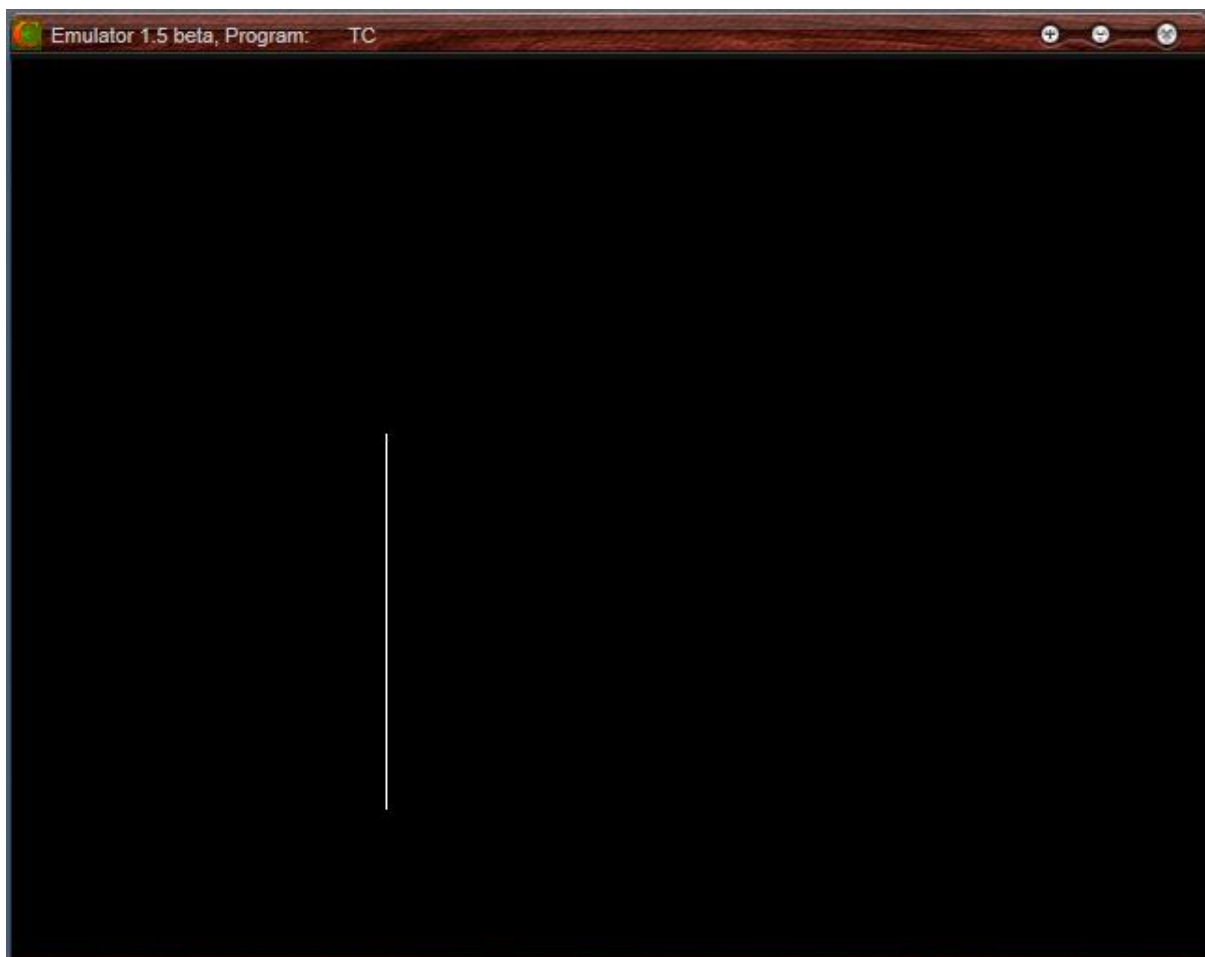
Choose any one line drawing algorithm:

1. DDA Algorithm
2. Bresenham Algorithm

Enter your choice: 1

Enter the start point of line (xa,ya): 200 200

Enter the end point of line (xb,yb): 200 400



Case 2:

Implementation of Algorithms for drawing 2D Primitives

Choose any one 2D Primitive:

1. Line
2. Circle

Enter your choice: 1

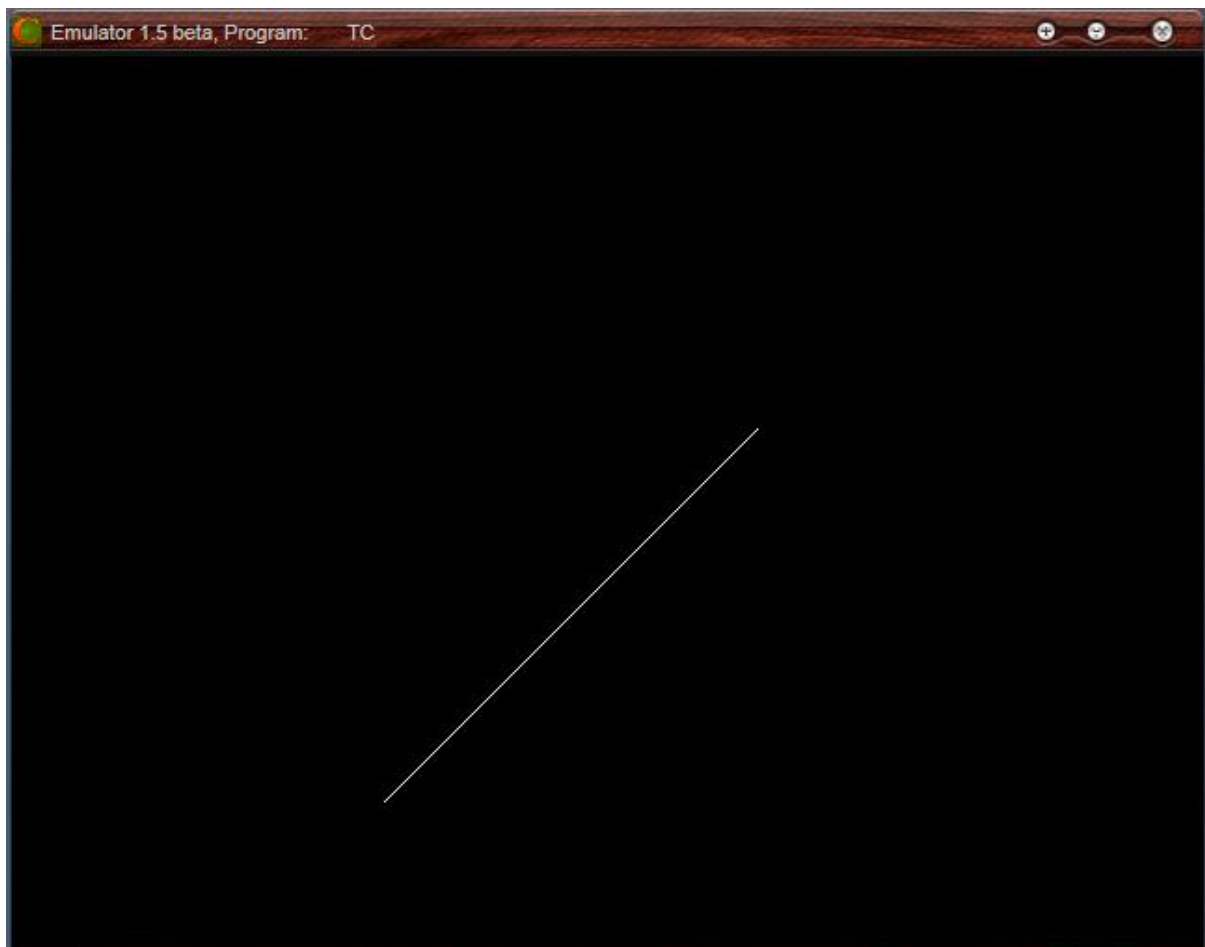
Choose any one line drawing algorithm:

1. DDA Algorithm
2. Bresenham Algorithm

Enter your choice: 1

Enter the start point of line (xa,ya): 400 200

Enter the end point of line (xb,yb): 200 400



Case 3:

Implementation of Algorithms for drawing 2D Primitives

Choose any one 2D Primitive:

1. Line
2. Circle

Enter your choice: 1

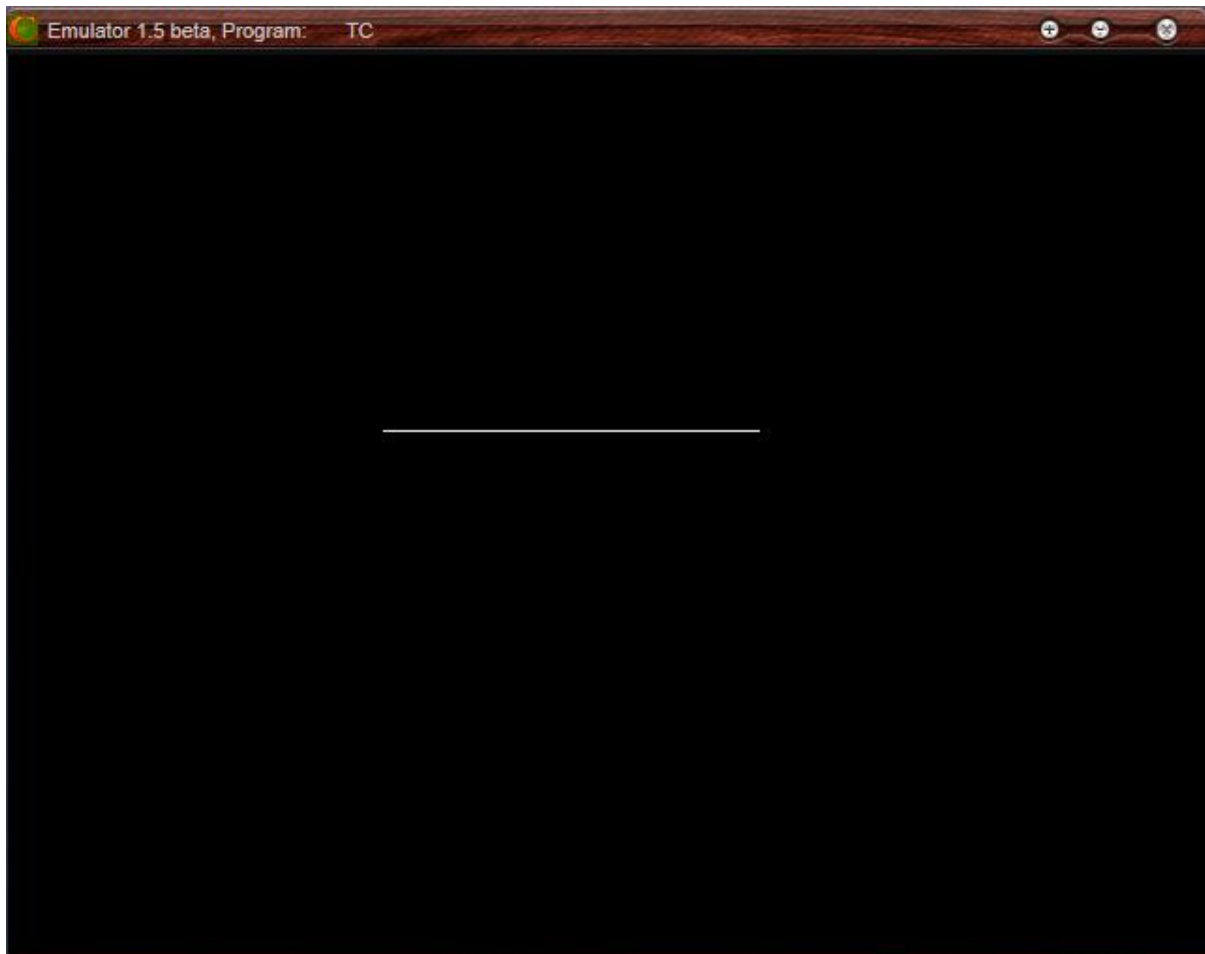
Choose any one line drawing algorithm:

1. DDA Algorithm
2. Bresenham Algorithm

Enter your choice: 2

Enter the start point of line (xa,ya): 200 200

Enter the end point of line (xb,yb): 400 200



Case 4:

Implementation of Algorithms for drawing 2D Primitives

Choose any one 2D Primitive:

1. Line
2. Circle

Enter your choice: 1

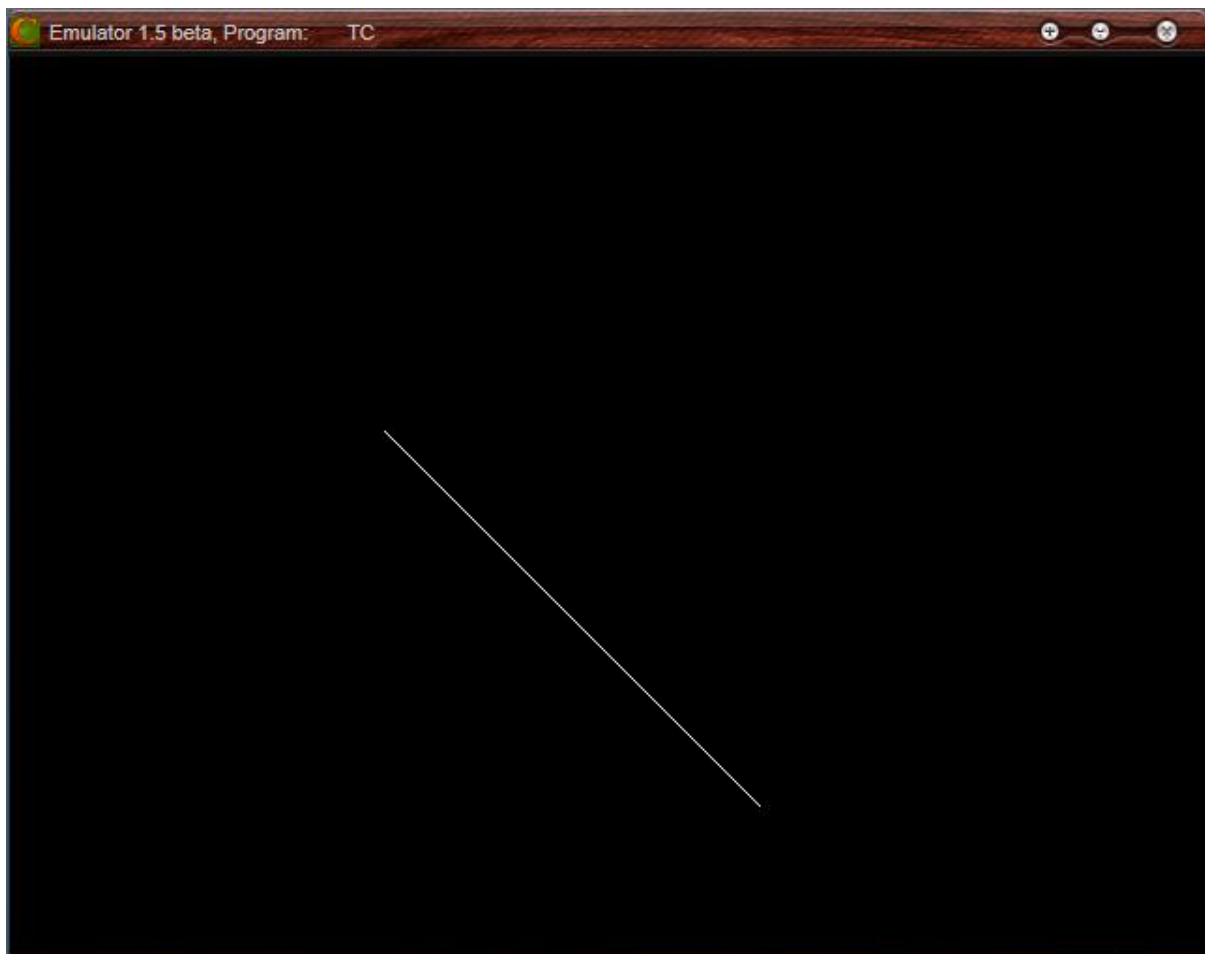
Choose any one line drawing algorithm:

1. DDA Algorithm
2. Bresenham Algorithm

Enter your choice: 2

Enter the start point of line (xa,ya): 200 200

Enter the end point of line (xb,yb): 400 400



Case 5:

Implementation of Algorithms for drawing 2D Primitives

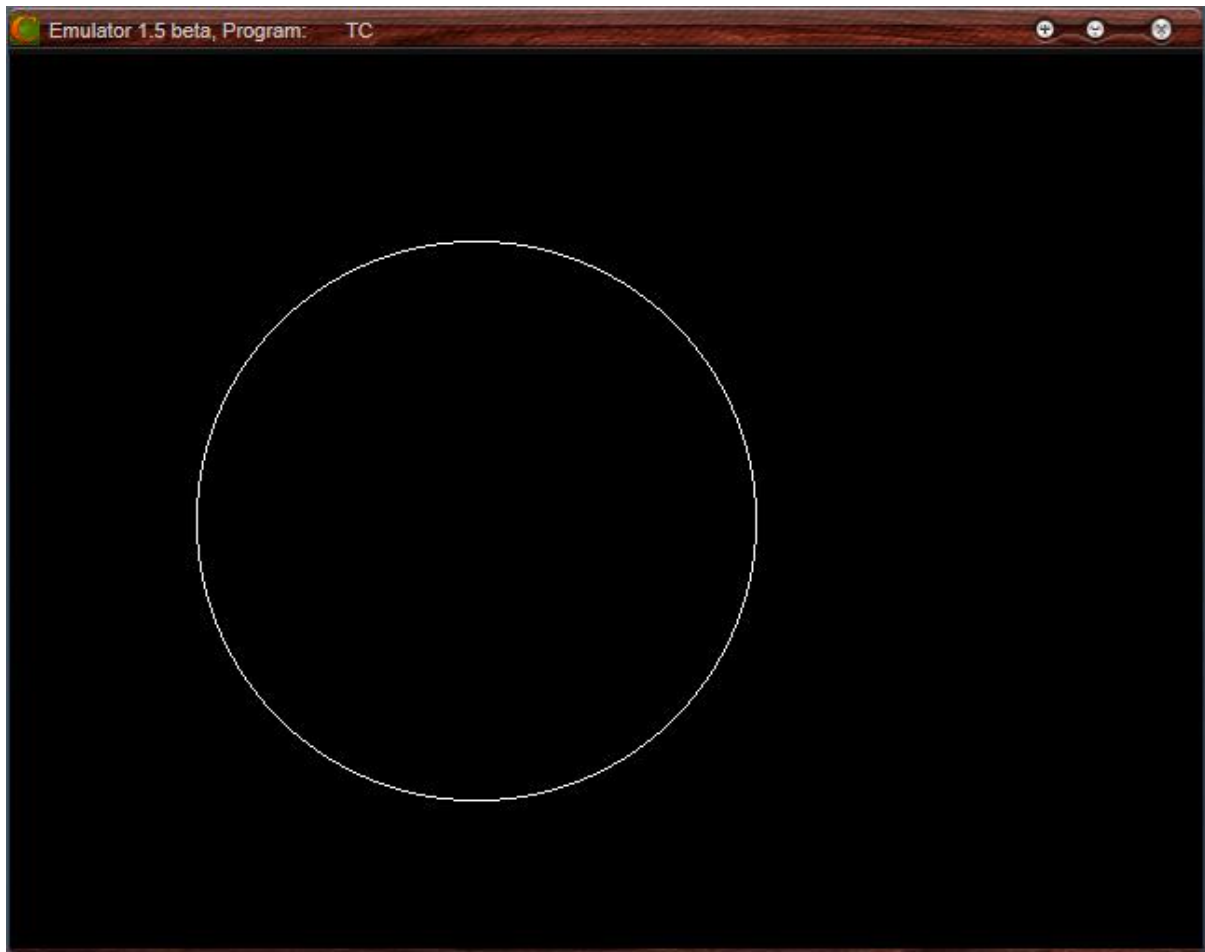
Choose any one 2D Primitive:

1. Line
2. Circle

Enter your choice: 2

Enter the center point of the circle: 250 250

Enter the radius of the circle: 150



Result:

Thus the implementation of the algorithms for drawing a line using DDA algorithm and Bresenham algorithm, and also drawing a circle using midpoint algorithm has been implemented and the output was verified.

Ex. No : 2

Date :

Implementation of 2D Geometric Transformations

Aim:

To implement the following 2D Geometric Transformations

- a. Translation
- b. Rotation
- c. Scaling
- d. Reflection
- e. Shearing
- f. Window-View Port

Algorithm:

1. Get the coordinates of triangle (x1, y1, x2, y2, x3, y3).
2. Draw the original triangle.
3. Print the menu for choosing 2D Geometric Transformation.
 - a. If user choose translation then get the translation factors(x, y) and draw the translated triangle in the following coordinates (x1+x, y1+y, x2+x, y2+y, x3+x, y3+y).
 - b. (i) If user choose rotation then get the rotation angle (t) and reference point of the rotation (rx, ry).
(ii) Change the t value to $t = t * (3.14 / 180)$ and calculate the rotated coordinates by the following formulae
$$rx1 = rx + (x1 - rx) * \cos(t) - (y1 - ry) * \sin(t);$$
$$ry1 = ry + (x1 - rx) * \sin(t) + (y1 - ry) * \cos(t);$$

(iii) Similarly calculate the coordinates rx2, ry2, rx3, ry3 and draw the rotated triangle in the following coordinates (rx1, ry1, rx2, ry2, rx3, ry3).
 - c. If user choose scaling then get the scaling factors(x, y) and draw the scaled triangle in the following coordinates (x1*x, y1*y, x2*x, y2*y, x3*x, y3*y).
 - d. If user choose reflection then rotate the triangle in 180° at (x2, y2) and draw the rotated triangle (which is reflected triangle).
 - e. If user choose shearing then get the shear value and draw the sheared triangle in the following coordinates (x1, y1, x2+x, y2, x3, y3).
 - f. (i) If user choose window-view port then draw the rectangle in window port coordinates (w1, w2, w3, w4) and draw the original triangle.
(ii) calculate the x, y and view port coordinates by following formulae
$$x = (v3 - v1) / (w3 - w1);$$
$$y = (v4 - v2) / (w4 - w2);$$
$$vx1 = v1 + \text{floor}(((x1 - w1) * x) + 0.5);$$
$$vy1 = v2 + \text{floor}(((y1 - w2) * y) + 0.5);$$

(iii) Similarly calculate the coordinates vx2, vy2, vx3, vy3
(iv) Draw the rectangle in view port coordinates (v1, v2, v3, v4) and draw the triangle in view port by the following coordinates (vx1, vy1, vx2, vy2, vx3, vy3)

Program:

```

#include <graphics.h>
#include <iostream.h>
#include <conio.h>
#include <math.h>
float x1, y1, x2, y2, x3, y3, x, y, rx1, ry1, rx2, ry2, rx3, ry3, t, vx1, vy1, vx2, vy2, vx3, vy3;
float w1 = 5, w2 = 5, w3 = 635, w4 = 465, v1 = 425, v2 = 75, v3 = 550, v4 = 250;
int gd, gm, ch;
void original ();
void triangle (float, float, float, float, float, float);
void rotate (float, float, float);
void main ()
{
    clrscr ();
    cout<< "\n\t ***** 2D Geometric Transformations *****";
    cout<< "\n\n Enter the coordinates of triangle (x1,y1,x2,y2,x3, y3): \n";
    cin>> x1 >> y1 >> x2 >> y2 >> x3 >> y3;
    original ();
    closegraph ();
    do
    {
        cout<< "\n\n Choose any one Transformation : ";
        cout<< "\n\t 1.Translation \n\t 2.Rotation \n\t 3.Scaling \n\t 4.Reflection";
        cout<< "\n\t 5.Shearing \n\t 6.Window-Viewport";
        cout<< "\n\n Enter your choice : \t";
        cin>>ch;
        switch (ch)
        {
            case 1:
                cout<< "\n Enter the translation factors (x,y): \t\t";
                cin>> x >> y;
                original ();
                triangle (x1+x, y1+y, x2+x, y2+y, x3+x, y3+y);
                closegraph ();
                break;
            case 2:
                cout<< "\n Enter the angle of rotation : \t\t";
                cin>> t;
                cout<< "\n Enter the reference point of rotation (rx,ry) : \t";
                cin>> x >> y;
                original ();
                rotate (t, x, y);
                closegraph ();
                break;
            case 3:

```

```

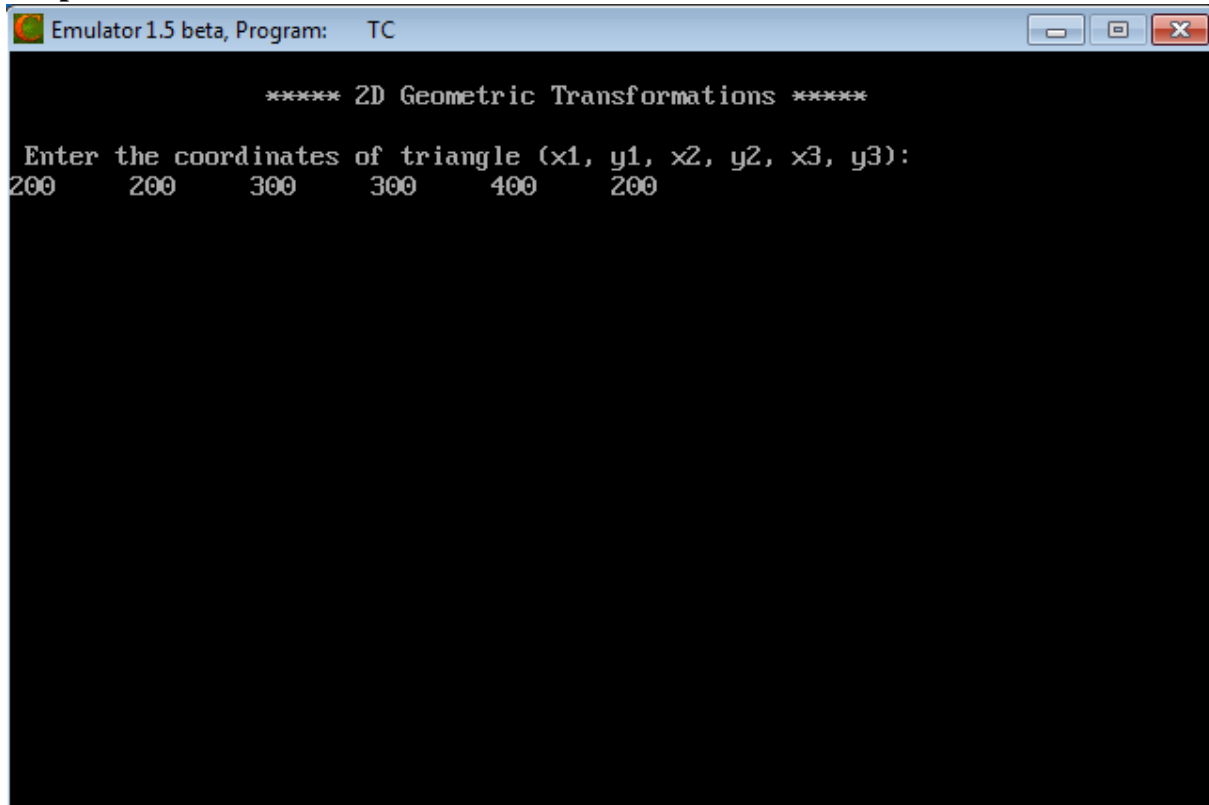
        cout<< "\n Enter the scaling factors (x,y): \t\t";
        cin>> x >> y;
        original ();
        triangle (x1*x, y1*y, x2*x, y2*y, x3*x, y3*y);
        closegraph ();
        break;
    case 4:
        original ();
        rotate (180, x2, y2);
        closegraph ();
        break;
    case 5:
        cout<< "\n Enter the Shear Value : \t\t";
        cin>> x;
        original ();
        triangle (x1, y1, x2+x, y2, x3, y3);
        closegraph ();
        break;
    case 6:
        initgraph (&gd, &gm, "C:\\TC\\BGI");
        rectangle (w1, w2, w3, w4);
        outtextxy (300, 10, "Window Port");
        triangle (x1, y1, x2, y2, x3, y3);
        x = (v3 - v1) / (w3 - w1);
        y = (v4 - v2) / (w4 - w2);
        vx1 = v1 + floor (((x1 - w1) * x) + 0.5);
        vy1 = v2 + floor (((y1 - w2) * y) + 0.5);
        vx2 = v1 + floor (((x2 - w1) * x) + 0.5);
        vy2 = v2 + floor (((y2 - w2) * y) + 0.5);
        vx3 = v1 + floor (((x3 - w1) * x) + 0.5);
        vy3 = v2 + floor (((y3 - w2) * y) + 0.5);
        rectangle (v1, v2, v3, v4);
        outtextxy (450, 85, "View Port");
        triangle (vx1, vy1, vx2, vy2, vx3, vy3);
        closegraph ();
    }
} while (ch<= 6);
getch ();
}
void original ()
{
    initgraph (&gd, &gm, "C:\\TC\\BGI");
    triangle (x1, y1, x2, y2, x3, y3);
}

void triangle (float x1, float y1, float x2, float y2, float x3, float y3)

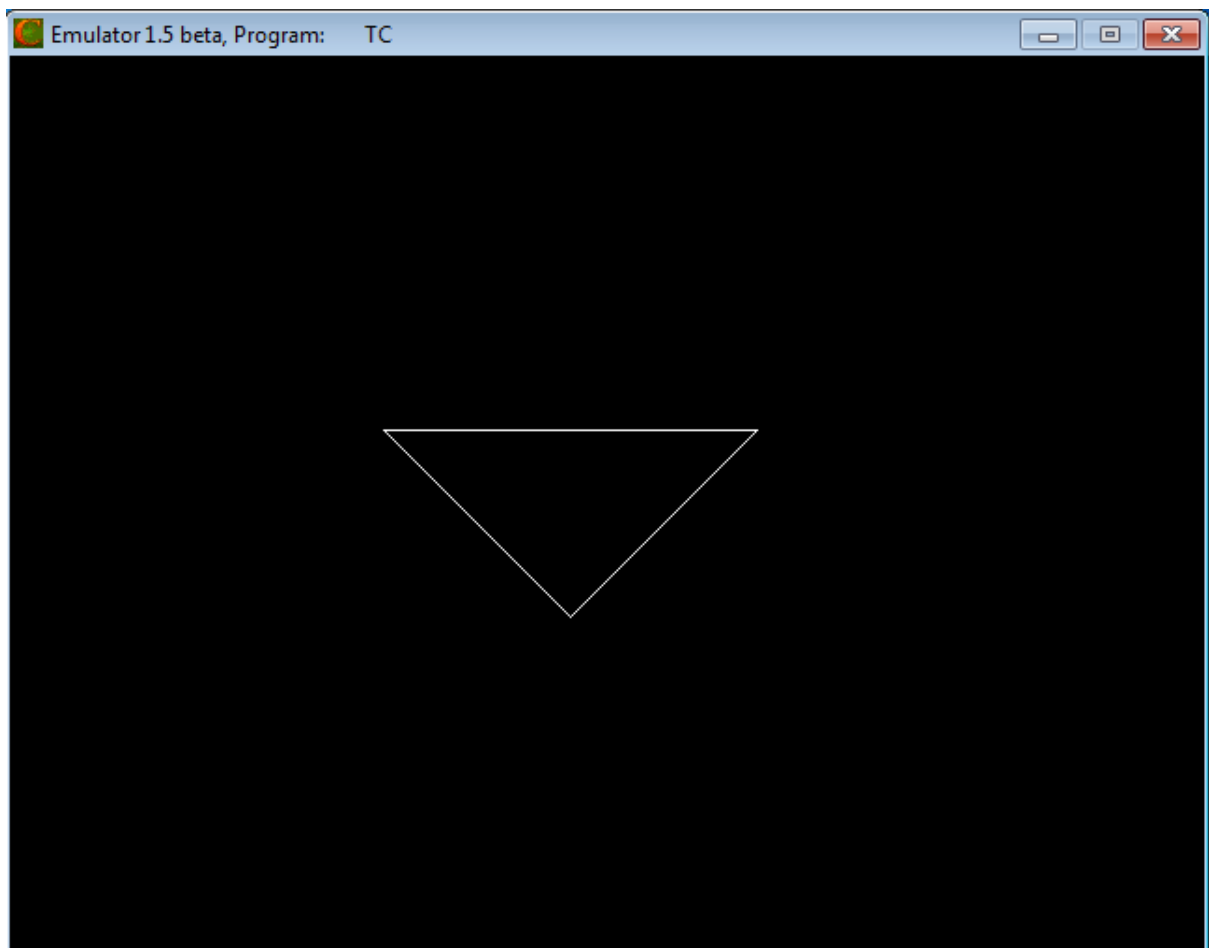
```

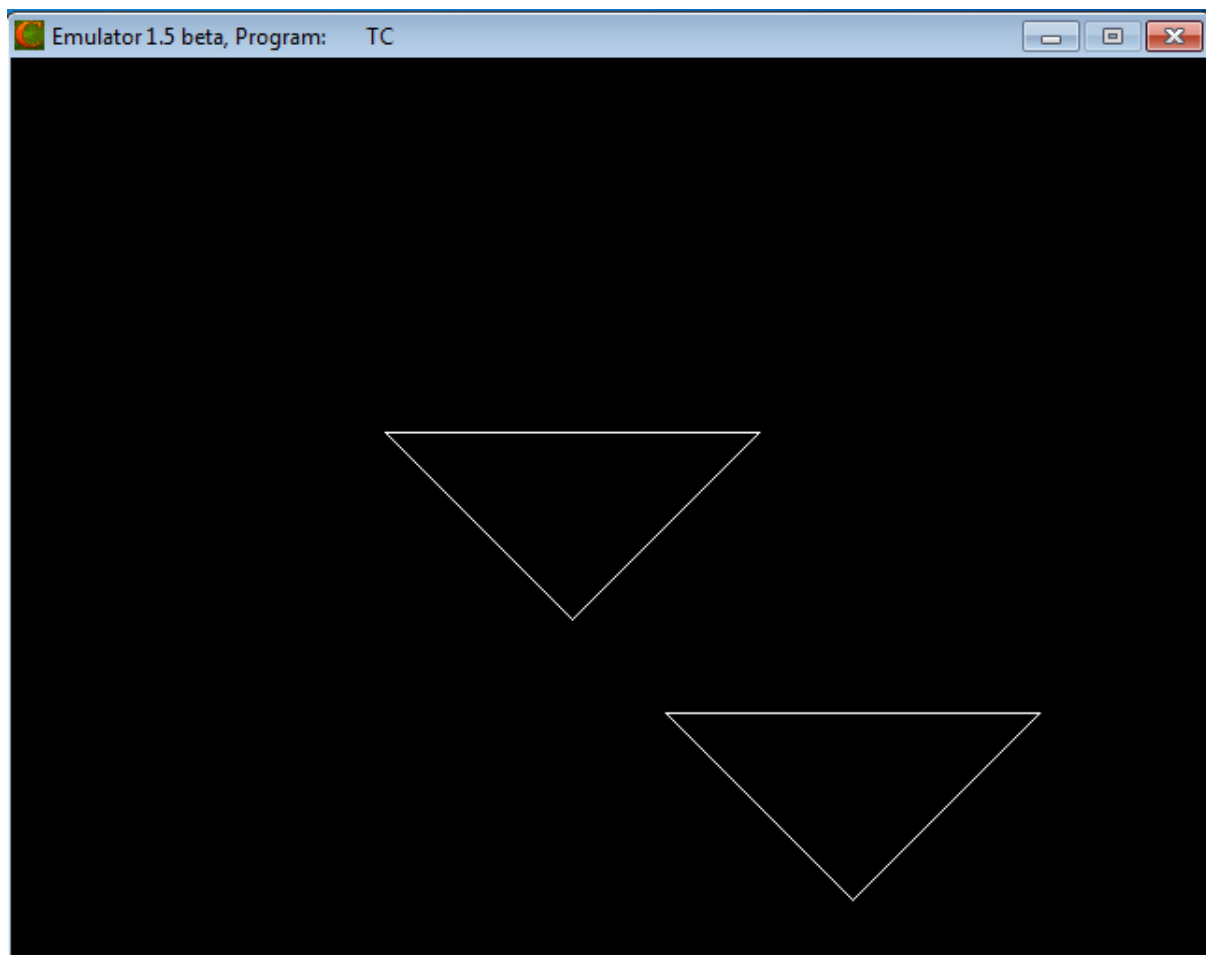
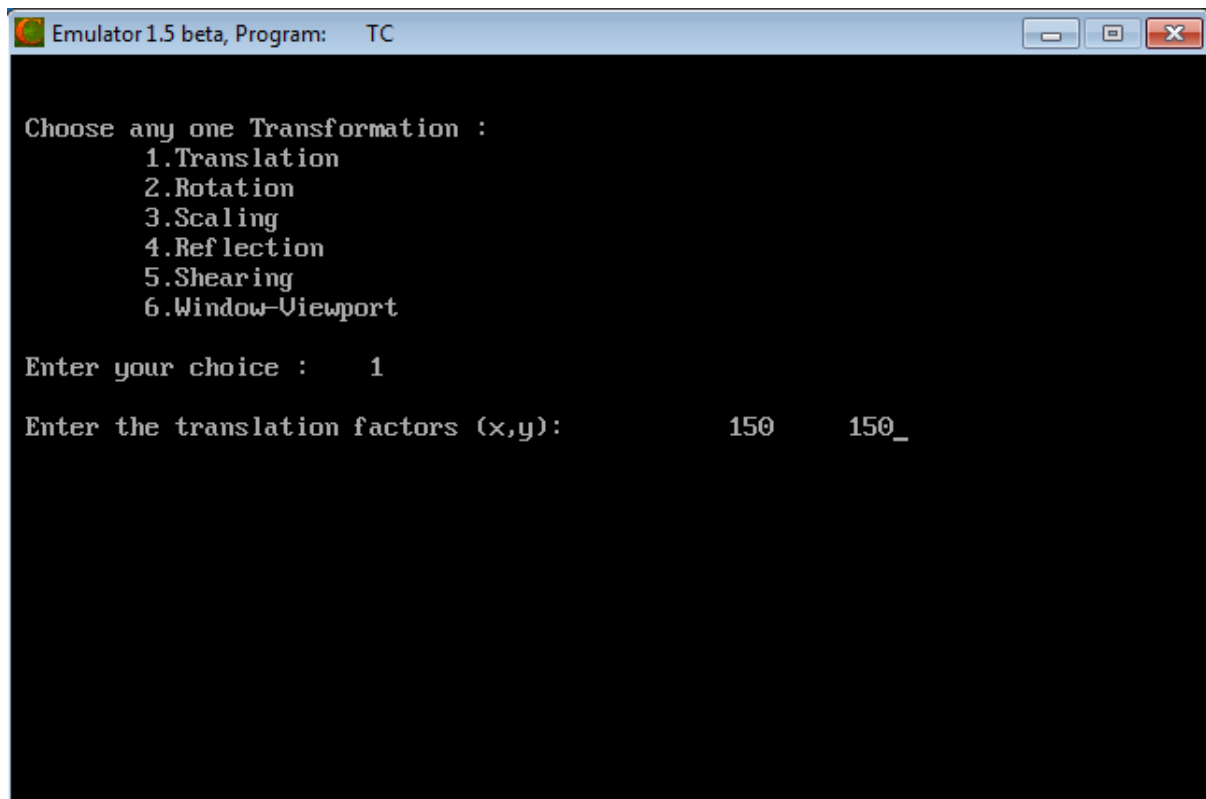
```
{  
    line (x1, y1, x2, y2);  
    line (x2, y2, x3, y3);  
    line (x3, y3, x1, y1);  
    getch ();  
}  
void rotate (float t, float rx, float ry)  
{  
    t = t * (3.14 / 180);  
    rx1 = rx + (x1 - rx) * cos (t) - (y1 - ry) * sin (t);  
    ry1 = ry + (x1 - rx) * sin (t) + (y1 - ry) * cos (t);  
    rx2 = rx + (x2 - rx) * cos (t) - (y2 - ry) * sin (t);  
    ry2 = ry + (x2 - rx) * sin (t) + (y2 - ry) * cos (t);  
    rx3 = rx + (x3 - rx) * cos (t) - (y3 - ry) * sin (t);  
    ry3 = ry + (x3 - rx) * sin (t) + (y3 - ry) * cos (t);  
    triangle (rx1, ry1, rx2, ry2, rx3, ry3);  
}
```

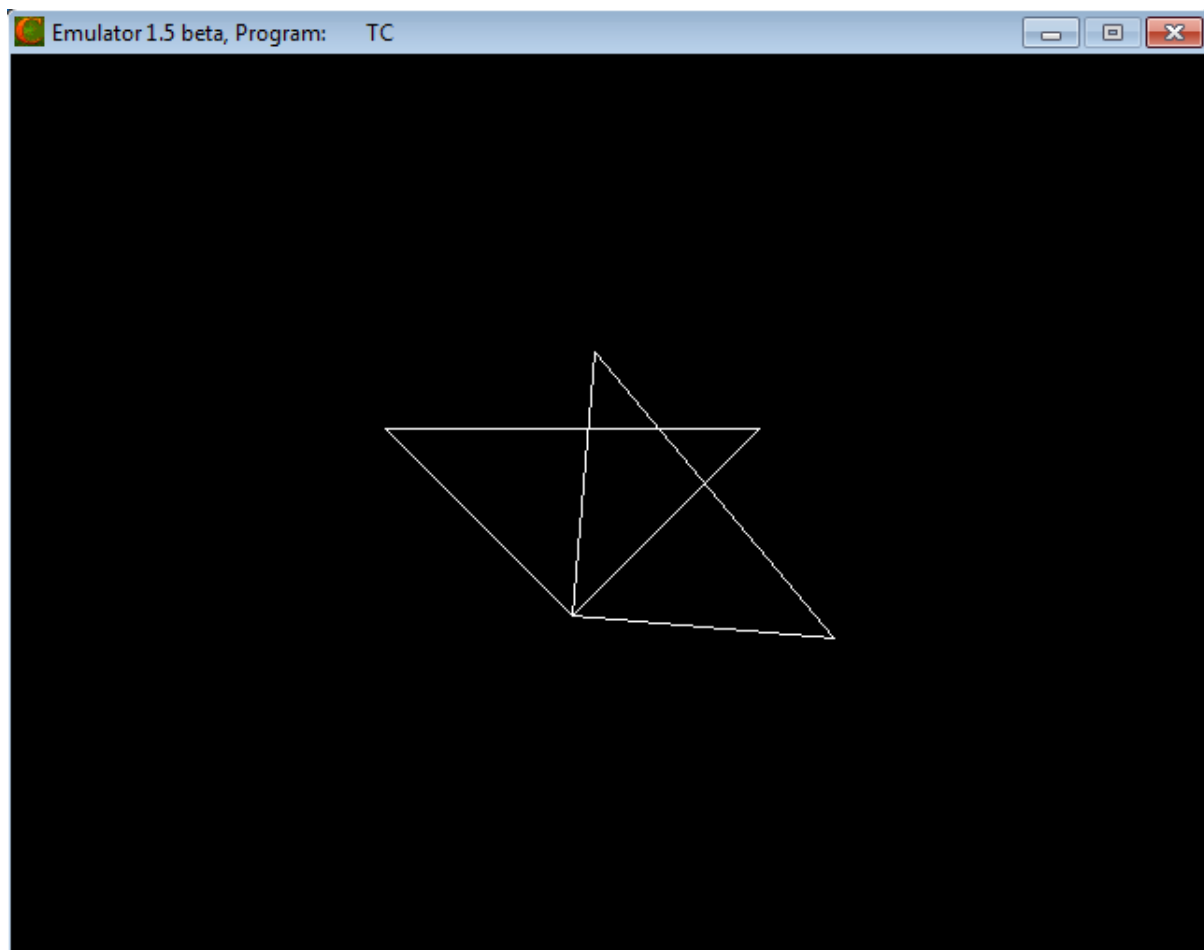
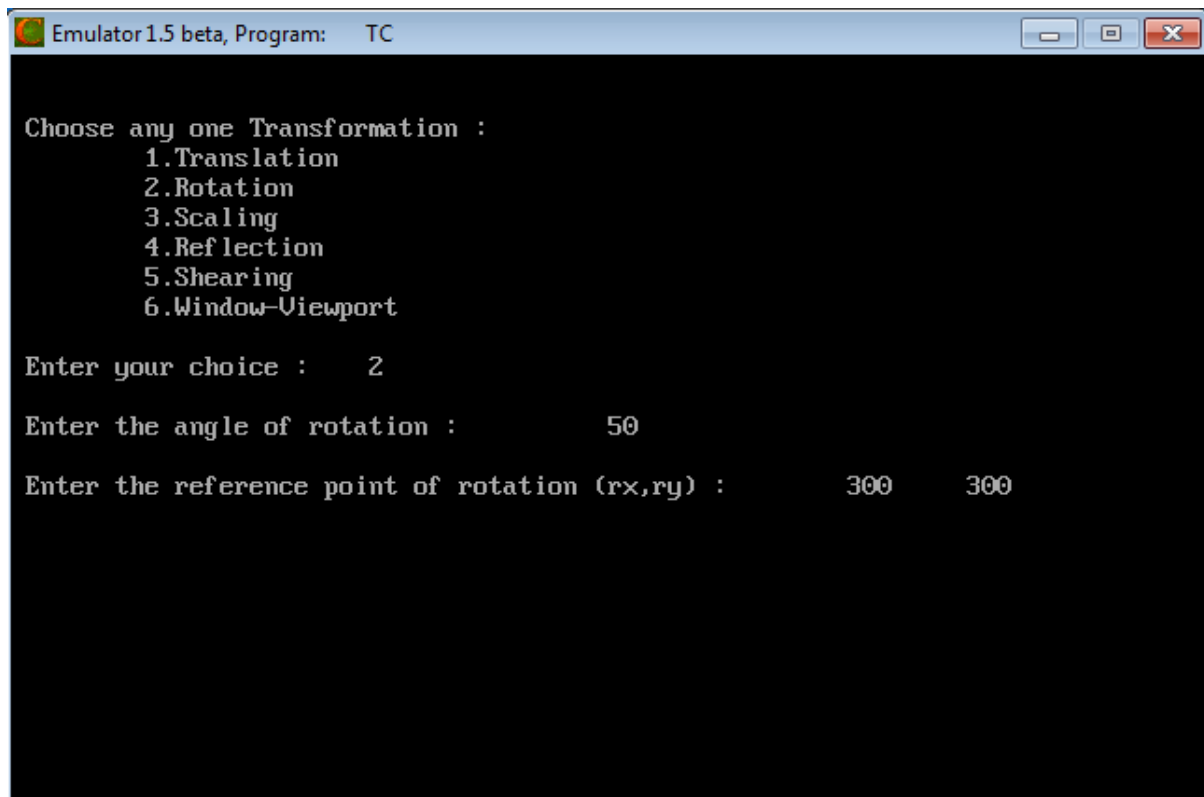
Output:

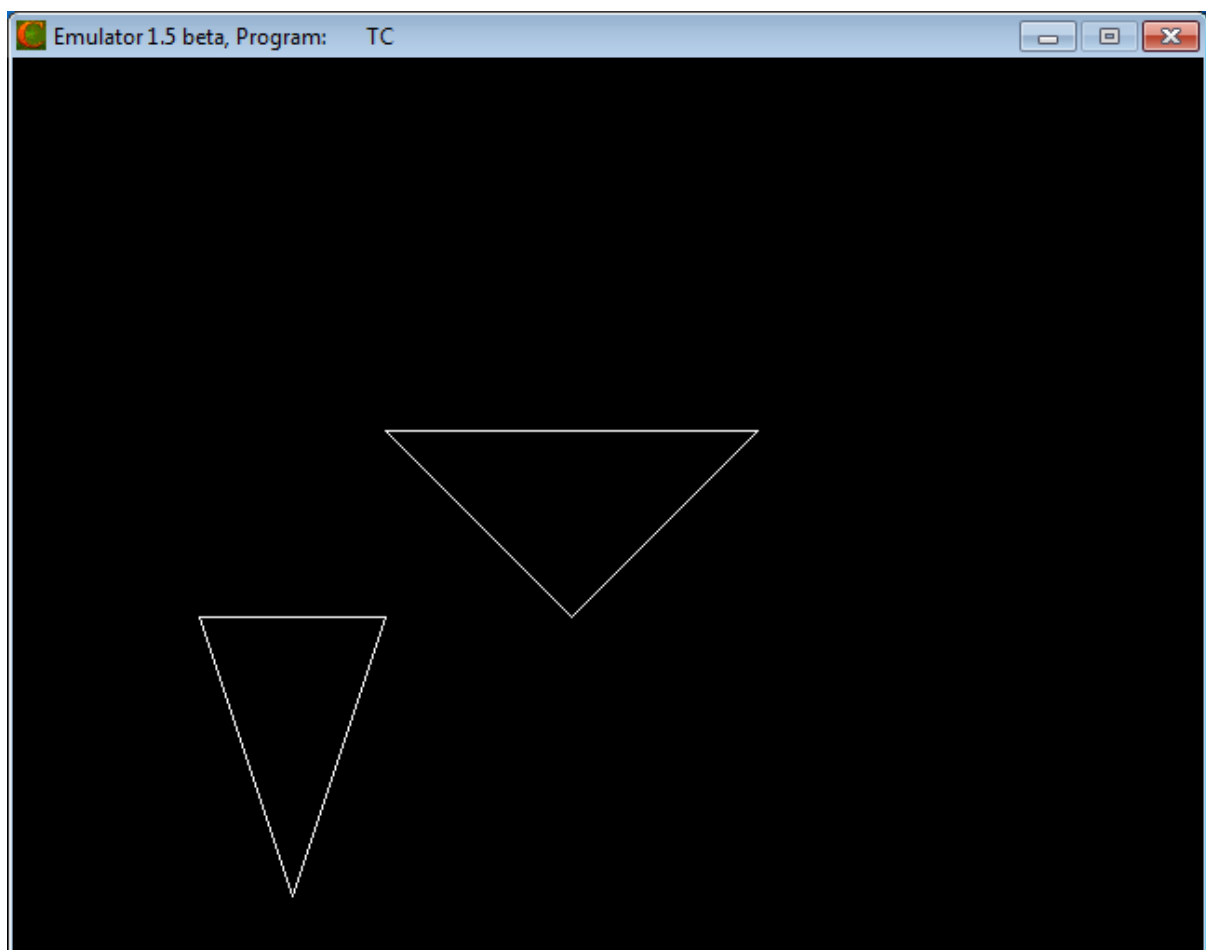
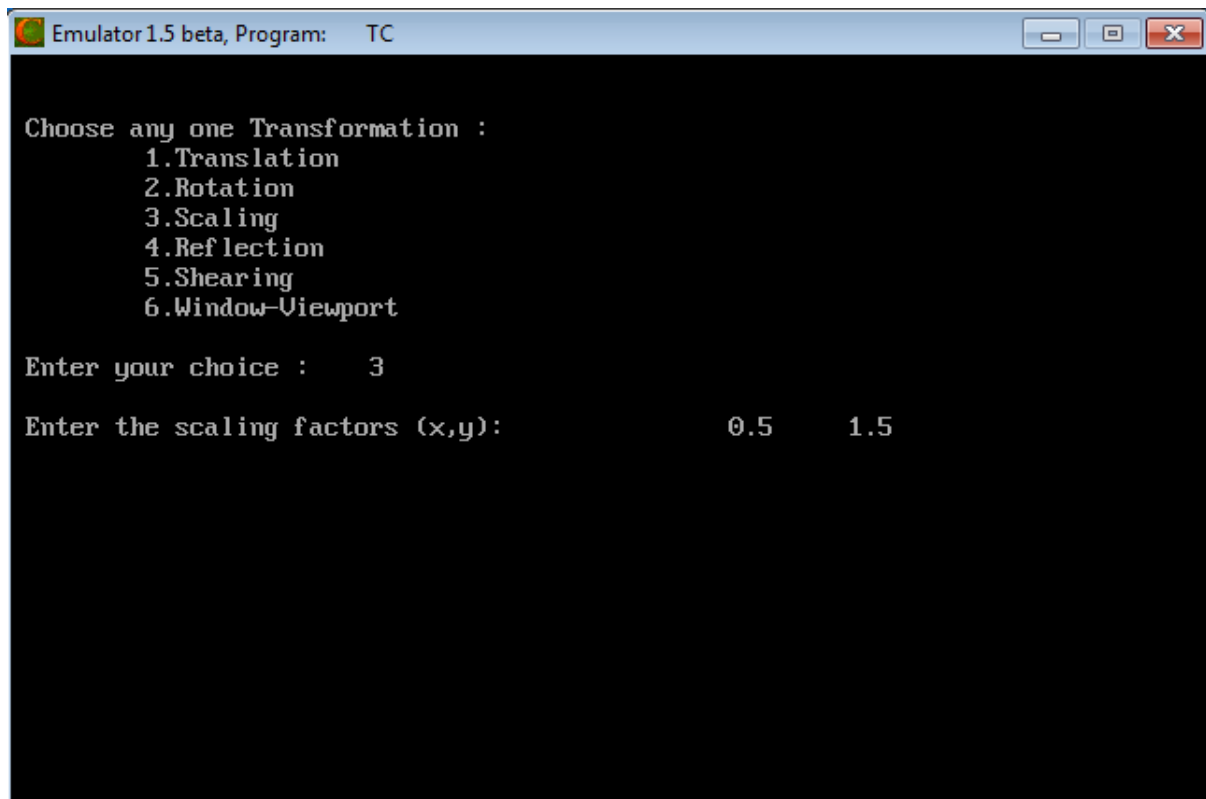


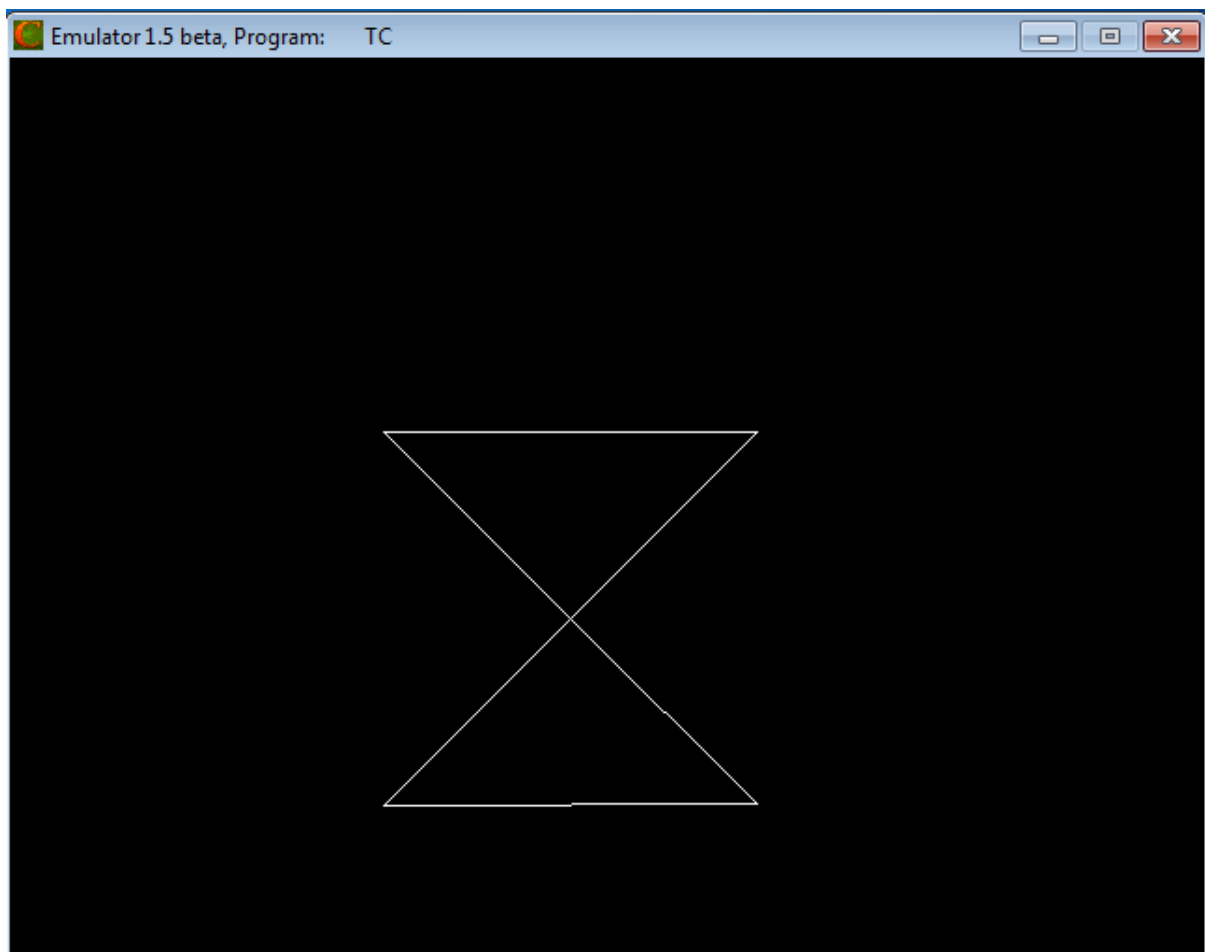
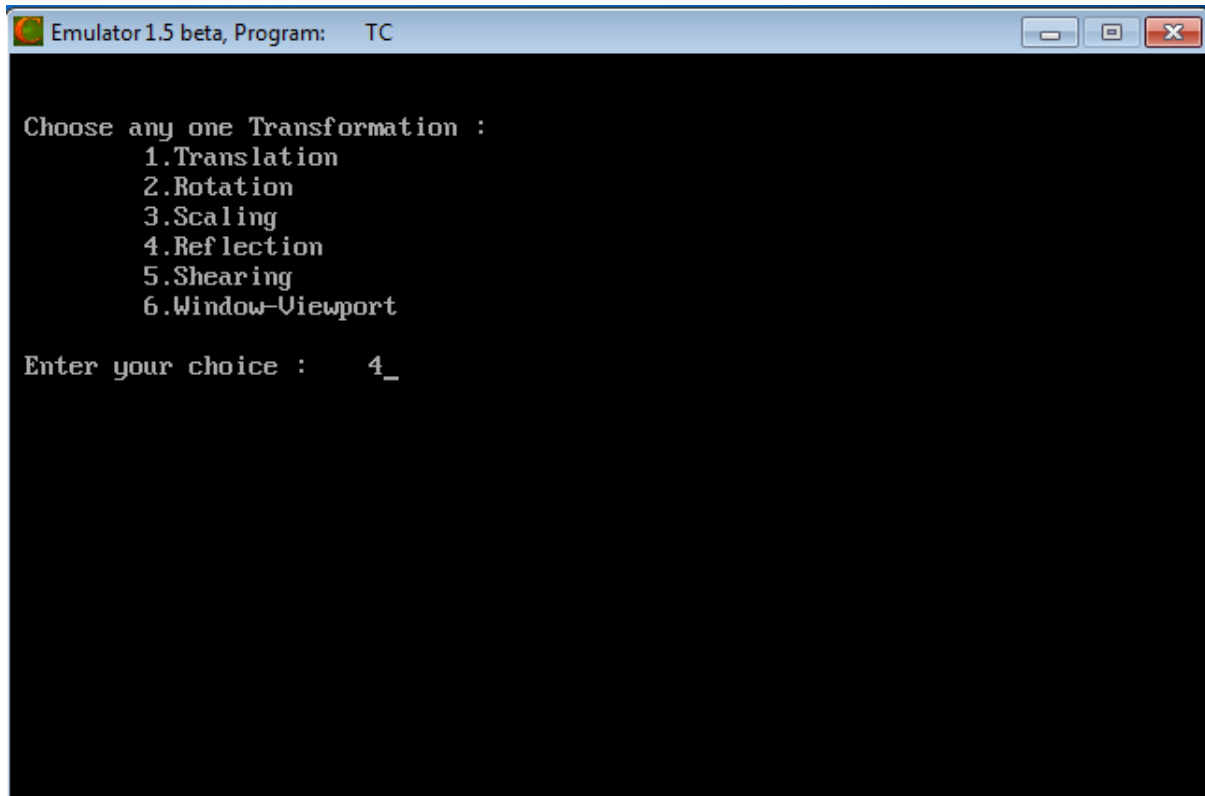
```
***** 2D Geometric Transformations *****  
Enter the coordinates of triangle (x1, y1, x2, y2, x3, y3):  
200      200      300      300      400      200
```

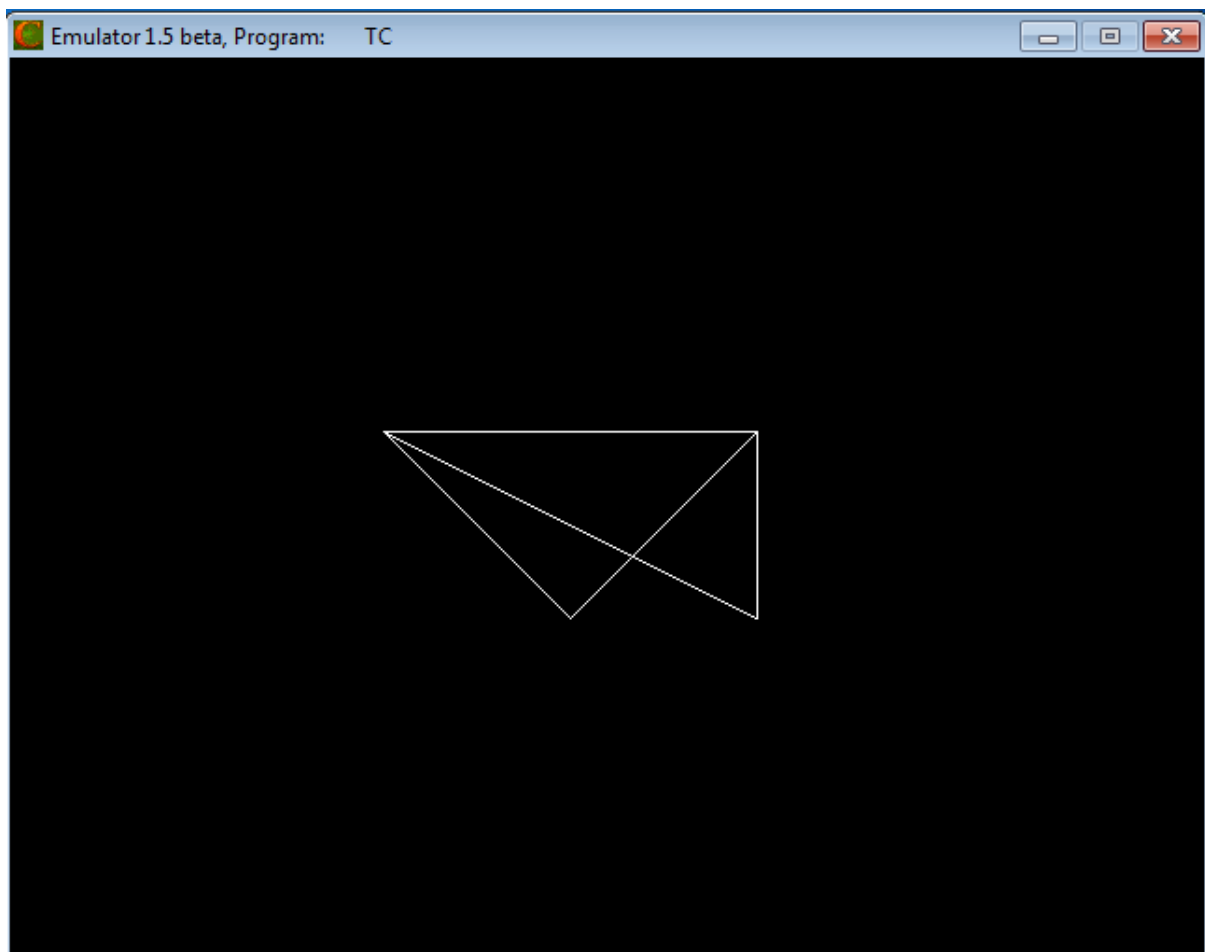
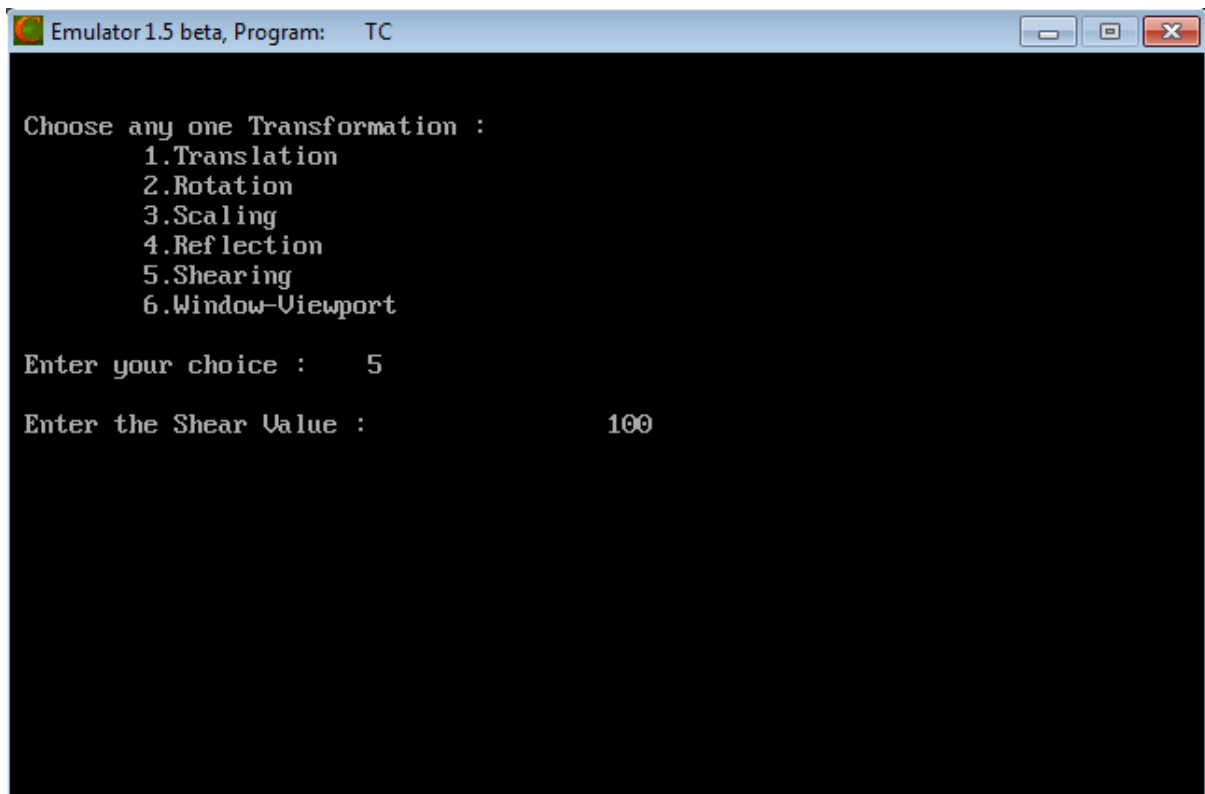




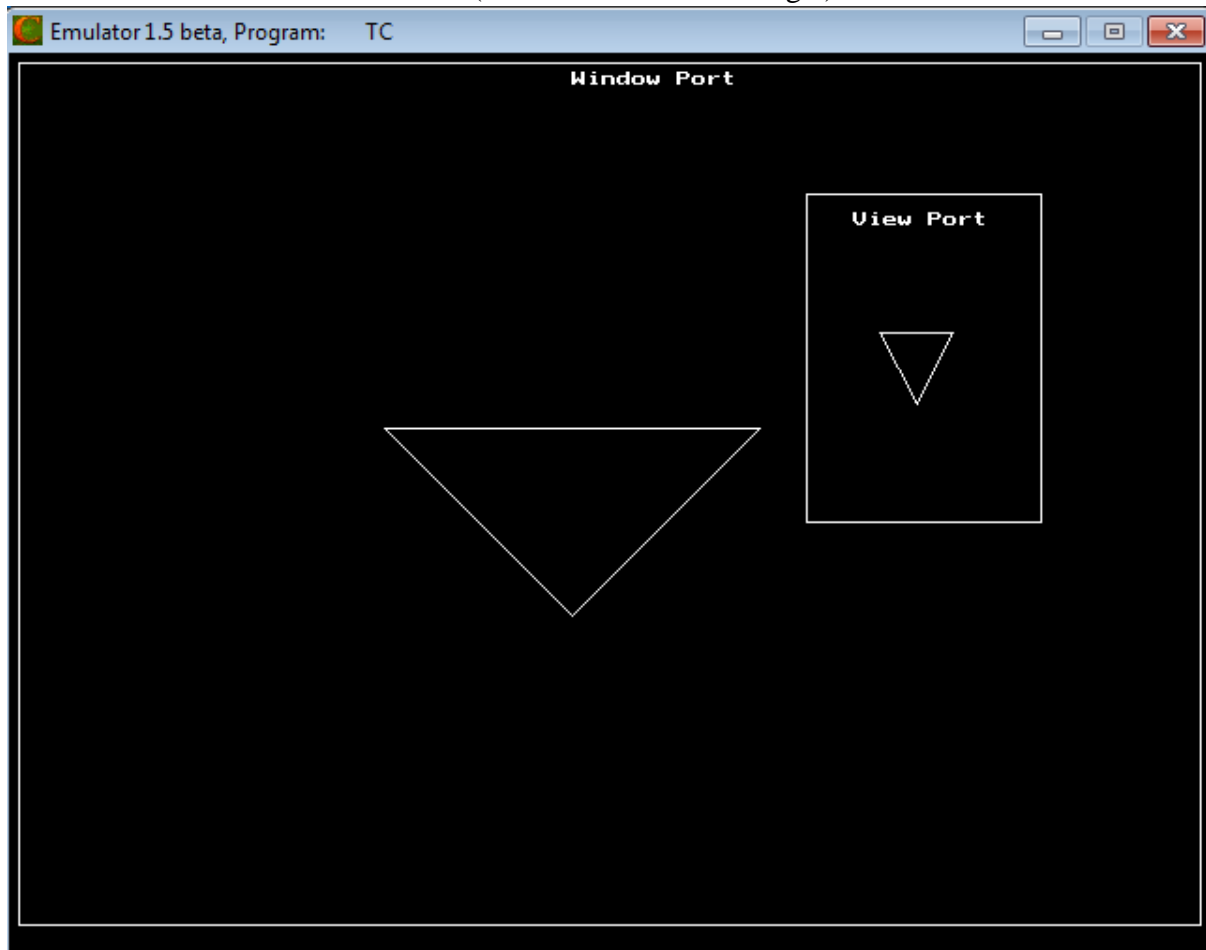








(If we choose 6 then we get)



(If we choose the number is in greater than 6 then we exit)

Result:

Thus the implementation of the 2D Geometric Transformations (translation, rotation, scaling, reflection, shearing, and window-view port) has been implemented and the output was verified.

Ex. No : 3

Date :

Implementation of Composite 2D Transformations

Aim:

To implement the following Composite 2D Transformations in the following order:

- a. Shearing
- b. Rotation
- c. Translation
- d. Reflection
- e. Scaling
- f. Window - View Port Transformation

Algorithm:

1. Get the coordinates of triangle (x1, y1, x2, y2, x3, y3).
2. Draw the original triangle.
3. Get the shear value and draw the sheared triangle in the following coordinates (x1, y1, x2+x, y2, x3, y3).
4. Get the rotation angle (t) and reference point of the rotation (rx, ry)
 - a) Change the t value to $t = t * (3.14 / 180)$ and calculate the rotated coordinates by the following formulae
$$rx1 = rx + (x1 - rx) * \cos(t) - (y1 - ry) * \sin(t);$$
$$ry1 = ry + (x1 - rx) * \sin(t) + (y1 - ry) * \cos(t);$$
 - b) Similarly calculate the coordinates rx2, ry2, rx3, ry3
 - c) Draw the rotated triangle in the following coordinates (rx1, ry1, rx2, ry2, rx3, ry3).
5. Get the translation factors(x, y) and draw the translated triangle in the following coordinates (x1+x, y1+y, x2+x, y2+y, x3+x, y3+y).
6. Rotate the triangle in 180^0 at (x2, y2) and draw the rotated triangle (which is reflected triangle).
7. Get the scaling factors(x, y) and draw the scaled triangle in the following coordinates (x1*x, y1*y, x2*x, y2*y, x3*x, y3*y).
8. (i) Draw the rectangle in window port coordinates (w1, w2, w3, w4) and draw the original triangle.
(ii) Calculate the x, y and view port coordinates by following formulae:
$$x = (v3 - v1) / (w3 - w1);$$
$$y = (v4 - v2) / (w4 - w2);$$
$$vx1 = v1 + \text{floor}(((x1 - w1) * x) + 0.5);$$
$$vy1 = v2 + \text{floor}(((y1 - w2) * y) + 0.5);$$

(iii) Similarly calculate the coordinates vx2, vy2, vx3, vy3.
(iv) Draw the rectangle in view port coordinates (v1, v2, v3, v4) and draw the triangle in view port by the following coordinates (vx1, vy1, vx2, vy2, vx3, vy3).

Program:

```

#include<graphics.h>
#include<iostream.h>
#include<conio.h>
#include<math.h>
float x1,y1,x2,y2,x3,y3,x,y,rx1,ry1,rx2,ry2,rx3,ry3,rx,ry,t,w1=5,w2=5,w3=635,w4=465,v1=425,
v2=75,v3=550,v4=250;
int gd,gm,i;
void triangle(float,float,float,float,float,float);
void rotate(float,float,float);
void main()
{
    clrscr();
    cout<<"\n\t***** Composite 2D Transformations *****";
    cout<<"\n\nEnter the coordinates of triangle (x1,y1,x2,y2,x3,y3) : ";
    cin>>x1>>y1>>x2>>y2>>x3>>y3;
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    triangle(x1,y1,x2,y2,x3,y3);
    closegraph();
    cout<<"\n\nEnter the Shear Value : \t\t";
    cin>>x;
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    triangle(x1,y1,x2,y2,x3,y3);
    x2=x2+x;
    triangle(x1,y1,x2,y2,x3,y3);
    closegraph();
    cout<<"\n\nEnter the angle of rotation : \t\t";
    cin>>t;
    cout<<"\n\nEnter the reference point of rotation (rx,ry) : \t";
    cin>>rx>>ry;
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    triangle(x1,y1,x2,y2,x3,y3);
    rotate(t,rx,ry);
    closegraph();
    cout<<"\n\nEnter the translation factors (x,y): \t\t";
    cin>>x>>y;
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    triangle(x1,y1,x2,y2,x3,y3);
    x1=x1+x;
    y1=y1+y;
    x2=x2+x;
    y2=y2+y;
    x3=x3+x;
    y3=y3+y;
    triangle(x1,y1,x2,y2,x3,y3);
    closegraph();
}

```

```

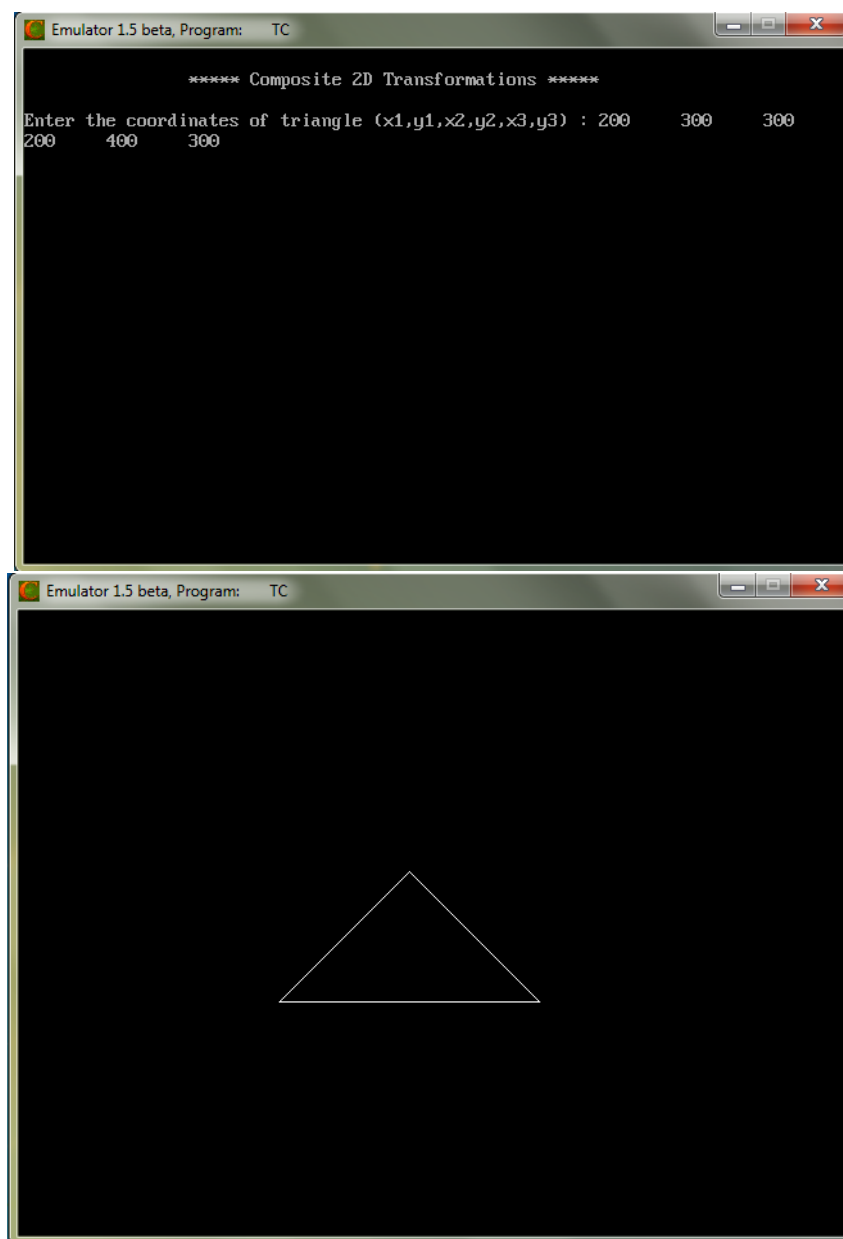
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    triangle(x1,y1,x2,y2,x3,y3);
    rotate(180,x2,y2);
    closegraph();
    cout<<"\n\nEnter the scaling factors (x,y): \t\t";
    cin>>x>>y;
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    triangle(x1,y1,x2,y2,x3,y3);
    x1=x1*x;
    y1=y1*y;
    x2=x2*x;
    y2=y2*y;
    x3=x3*x;
    y3=y3*y;
    triangle(x1,y1,x2,y2,x3,y3);
    closegraph();
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    rectangle(w1,w2,w3,w4);
    outtextxy(300,10,"Window Port");
    triangle(x1,y1,x2,y2,x3,y3);
    x=(v3-v1)/(w3-w1);
    y=(v4-v2)/(w4-w2);
    x1=v1+floor(((x1-w1)*x)+0.5);
    y1=v2+floor(((y1-w2)*y)+0.5);
    x2=v1+floor(((x2-w1)*x)+0.5);
    y2=v2+floor(((y2-w2)*y)+0.5);
    x3=v1+floor(((x3-w1)*x)+0.5);
    y3=v2+floor(((y3-w2)*y)+0.5);
    rectangle(v1,v2,v3,v4);
    outtextxy(450,85,"View Port");
    triangle(x1,y1,x2,y2,x3,y3);
}
void triangle(float x1,float y1,float x2,float y2,float x3,float y3)
{
    line(x1,y1,x2,y2);
    line(x2,y2,x3,y3);
    line(x3,y3,x1,y1);
    getch();
}
void rotate(float t,float rx,float ry)
{
    t=t*(3.14/180);
    rx1=rx+(x1-rx)*cos(t)-(y1-ry)*sin(t);
    ry1=ry+(x1-rx)*sin(t)+(y1-ry)*cos(t);
    rx2=rx+(x2-rx)*cos(t)-(y2-ry)*sin(t);
    ry2=ry+(x2-rx)*sin(t)+(y2-ry)*cos(t);

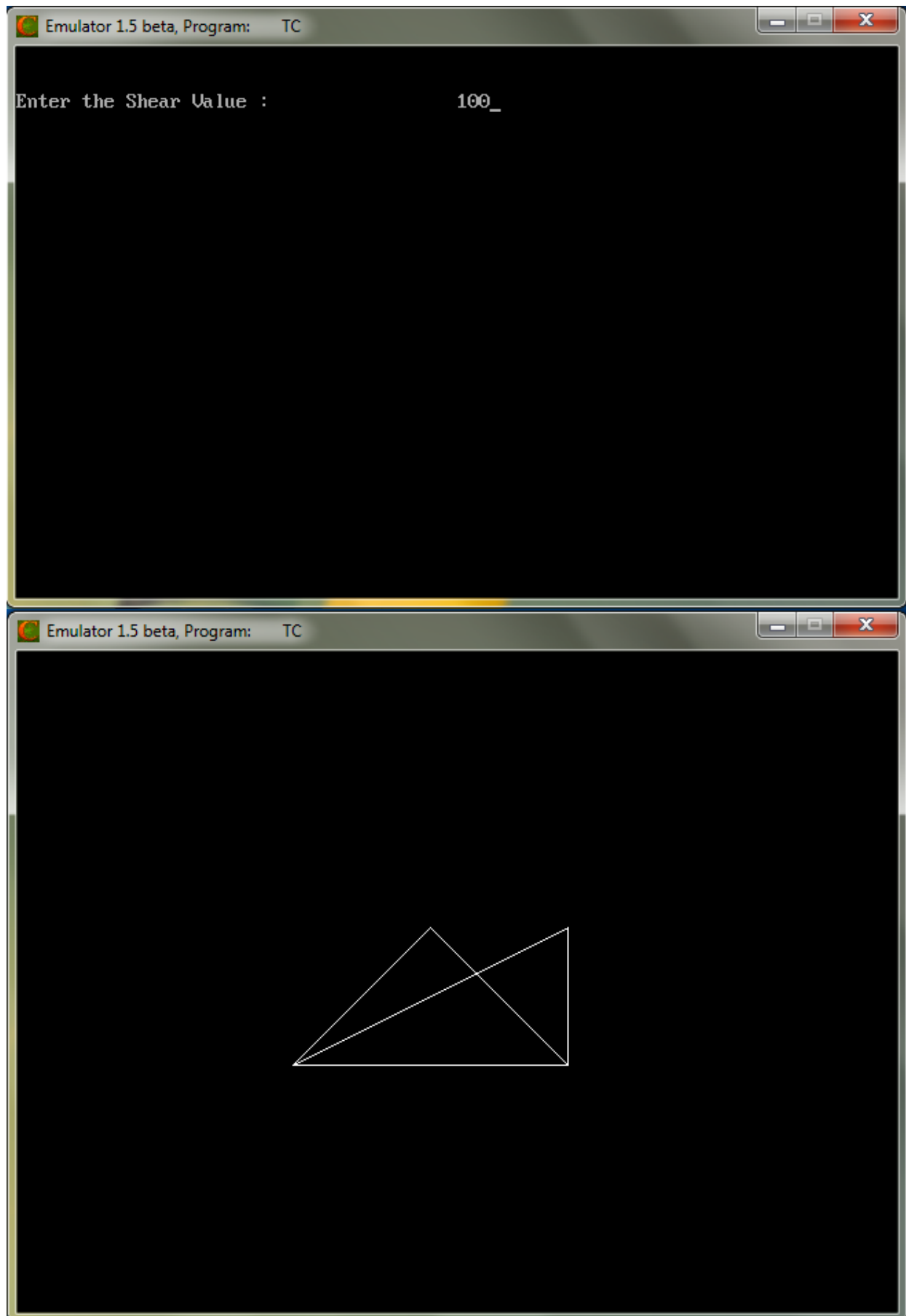
```

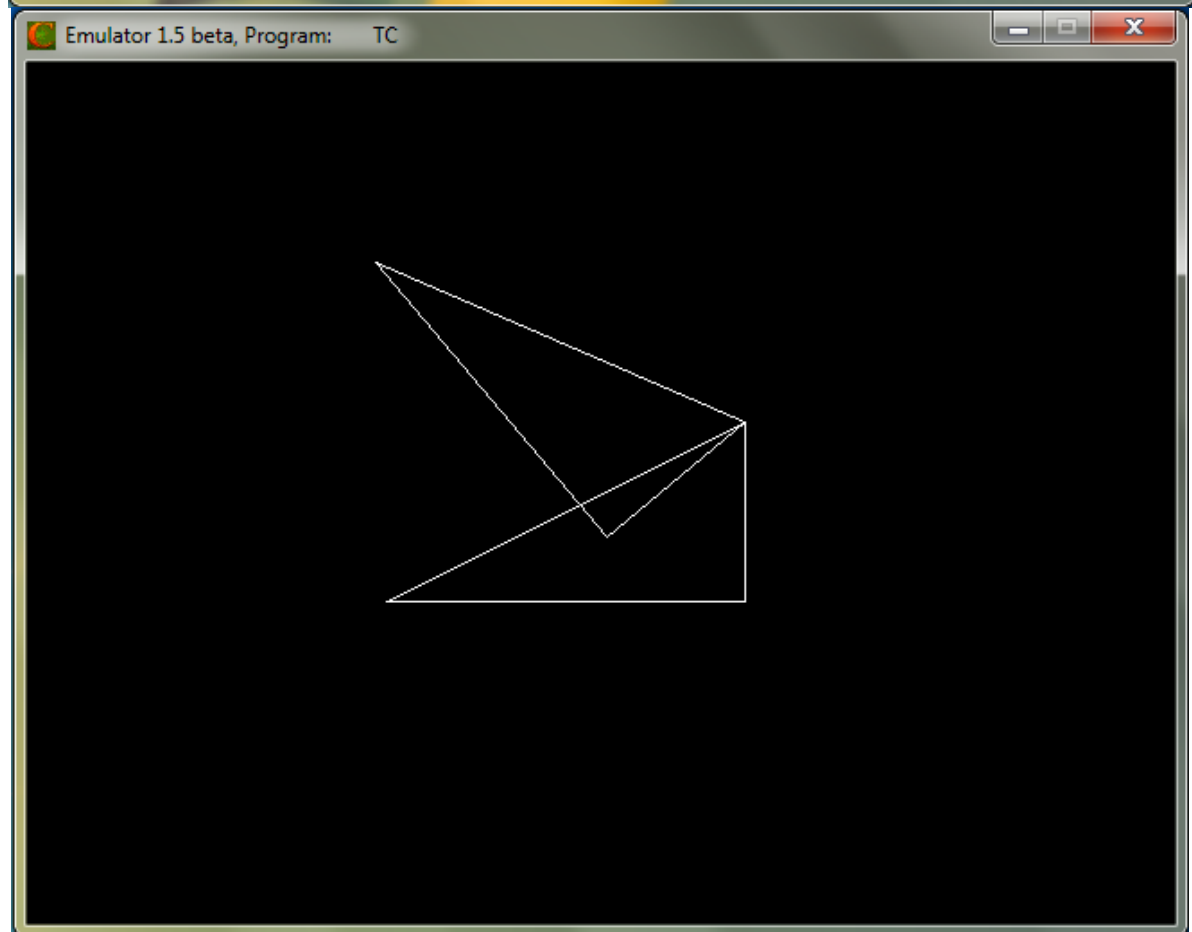
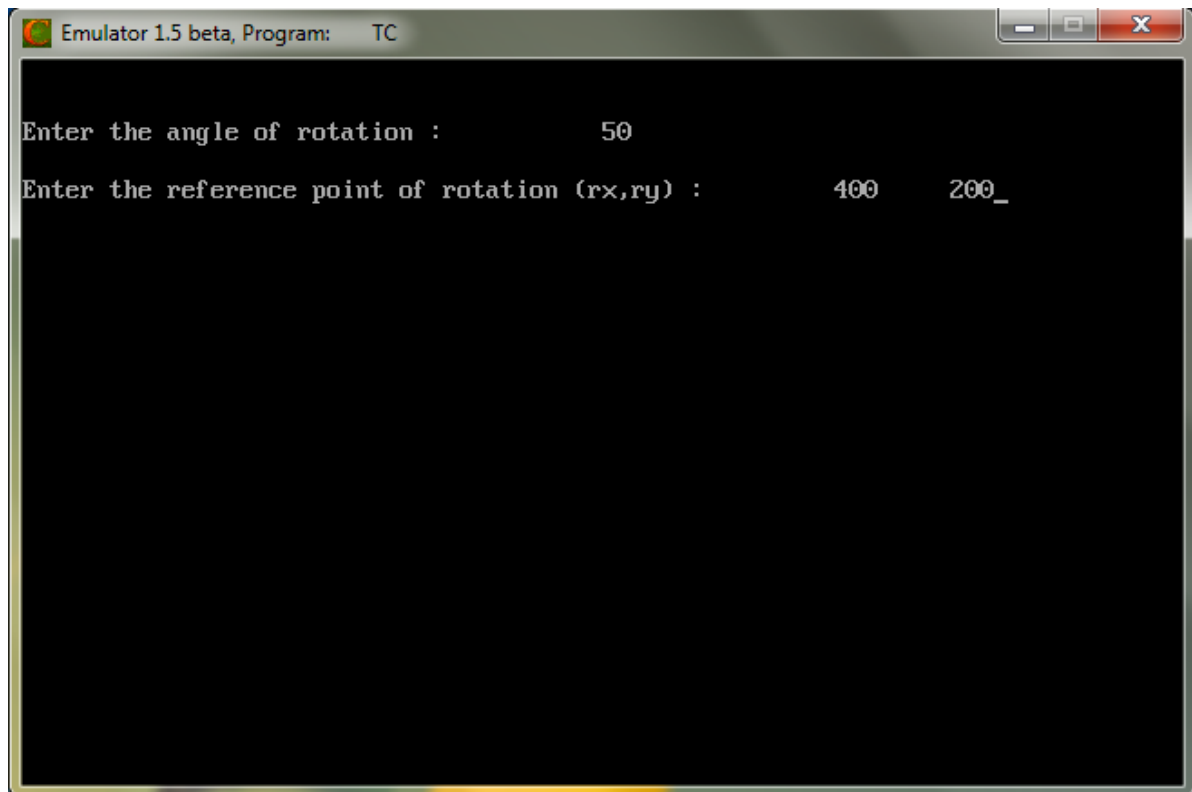

CS6513 COMPUTER GRAPHICS LABORATORY

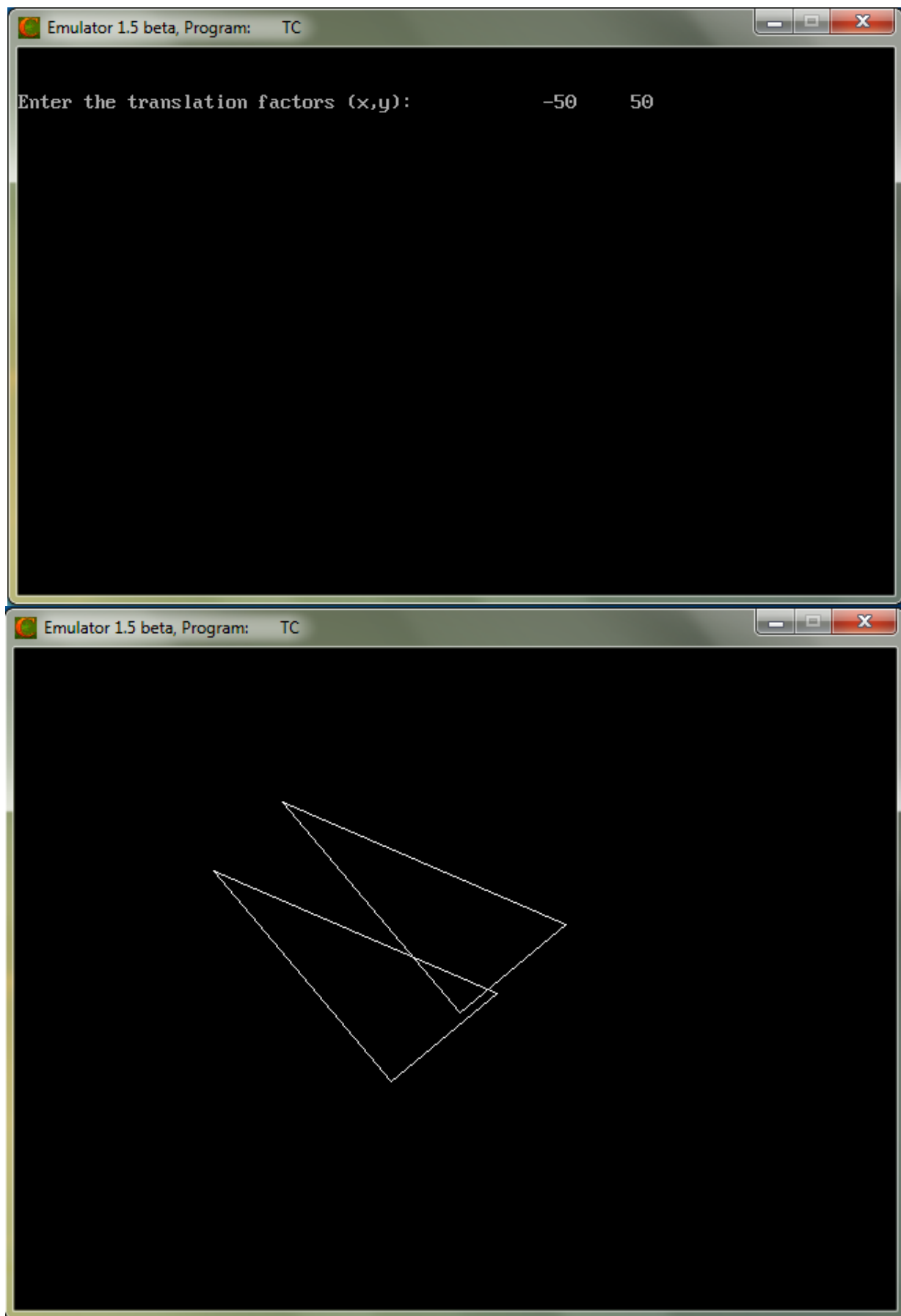
```
rx3=rx+(x3-rx)*cos(t)-(y3-ry)*sin(t);  
ry3=ry+(x3-rx)*sin(t)+(y3-ry)*cos(t);  
x1=rx1;  
y1=ry1;  
x2=rx2;  
y2=ry2;  
x3=rx3;  
y3=ry3;  
triangle(x1,y1,x2,y2,x3,y3);  
}
```

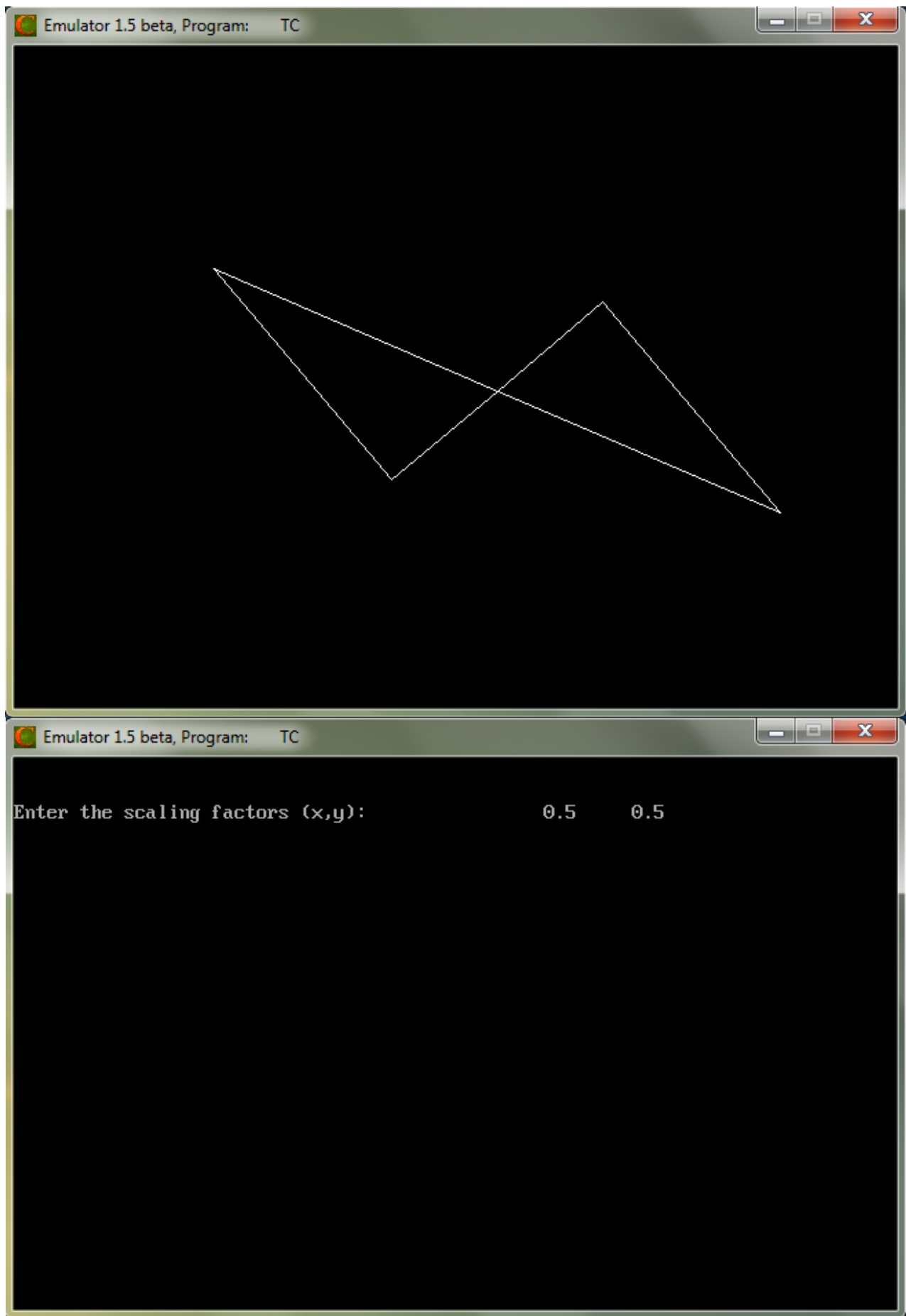
Output:

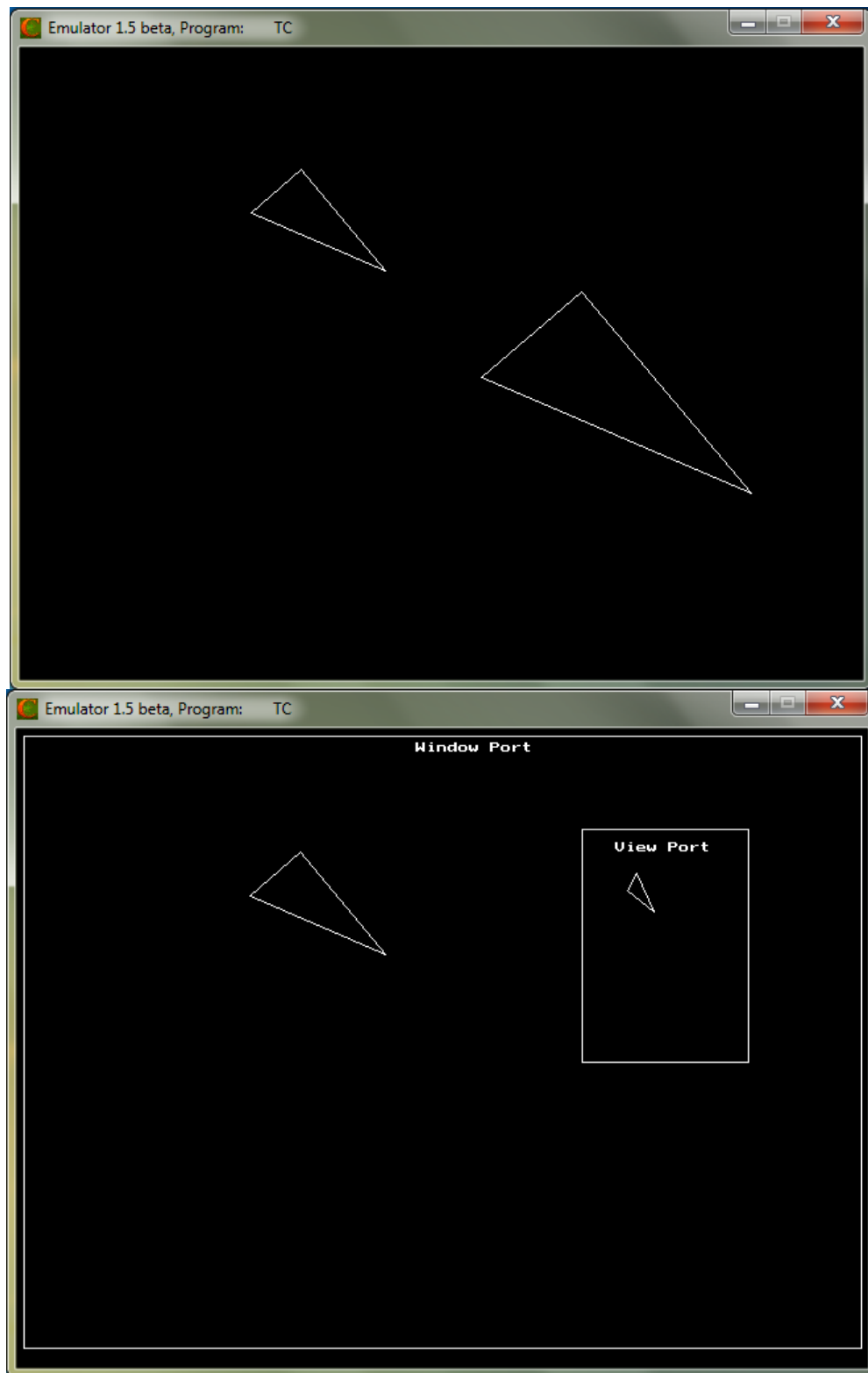












Result:

Thus the program to implementation of composite 2D transformations has been implemented and the output was verified.

Ex. No : 4

Date :

Line Clipping

Aim:

To write a program to implement the line clipping.

Algorithm:

1. Get the clip window coordinates.
2. Get the line end points.
3. Draw the window and the line.
4. Remove the line points which are plotted in outside the window.
5. Draw the window with clipped line.

Program:

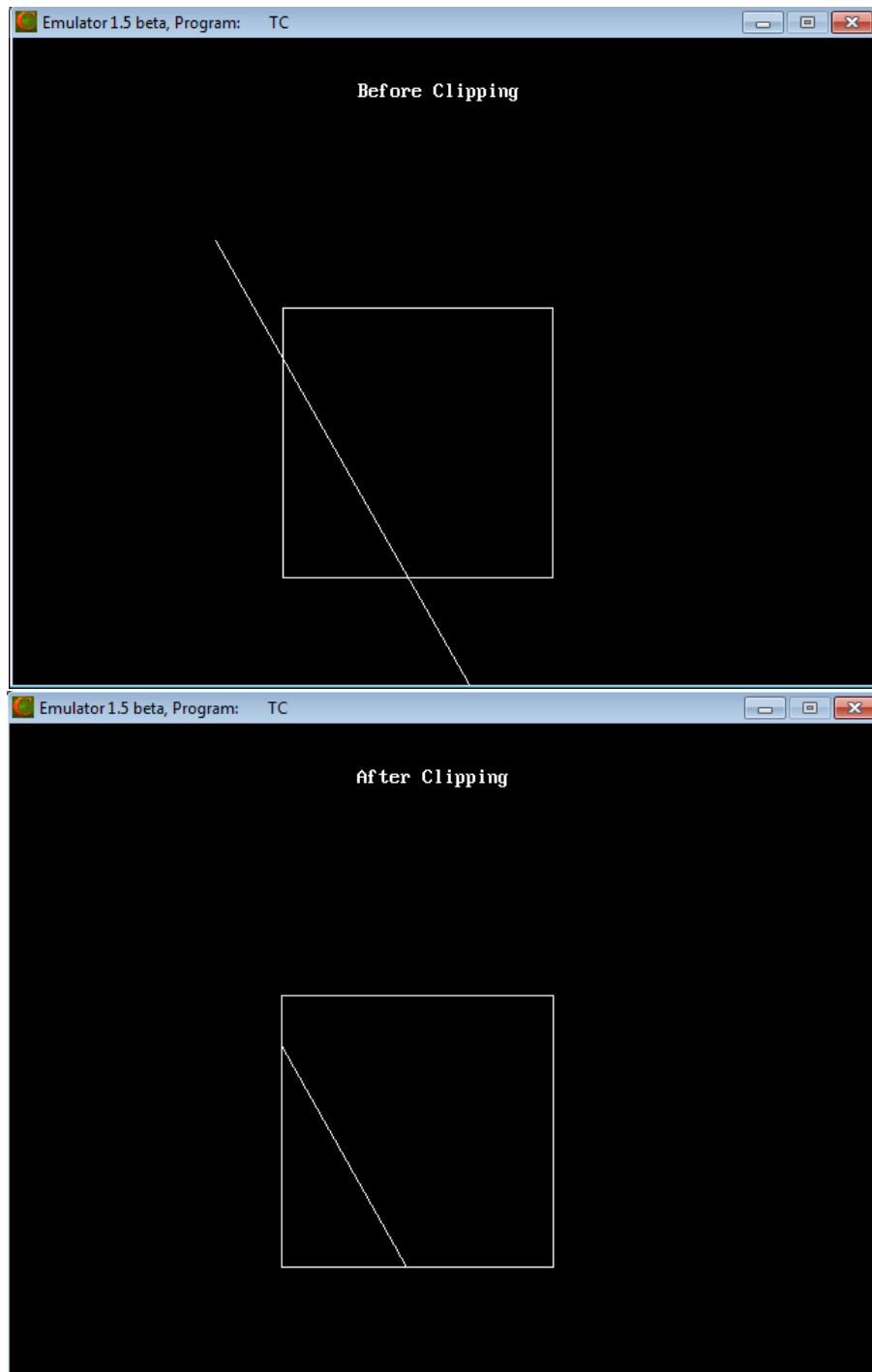
```
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
float x1,y1,x2,y2,wx1,wy1,wx2,wy2,m;
int gd,gm;
void main()
{
    clrscr();
    cout<<"\nEnter the clip window coordinates : ";
    cin>>wx1>>wy1>>wx2>>wy2;
    cout<<"\nEnter the line end points : ";
    cin>>x1>>y1>>x2>>y2;
    m=(y2-y1)/(x2-x1);
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    cout<<"\n\n\t\t\tBefore Clipping";
    rectangle(wx1,wy1,wx2,wy2);
    line(x1,y1,x2,y2);
    getch();
    closegraph();
    if(x1<wx1)
    {
        y1=y1+(wx1-x1)*m;
        x1=wx1;
    }
    if(x2>wx2)
    {
        y2=y2-(x2-wx2)*m;
```

```
        x2=wx2;
    }
    if(y2>wy2)
    {
        x2=x2-((y2-wy2)/m);
        y2=wy2;
    }
    if(y1<wy1)
    {
        x1=x1+(wy1-y1)/m;
        y1=wy1;
    }
    if(x2<wx1)
    {
        y2=y2+(wx1-x2)*m;
        x2=wx1;
    }
    if(x1>wx2)
    {
        y1=y1-(x1-wx2)*m;
        x1=wx2;
    }
    if(y2<wy1)
    {
        x2=x2+((wy1-y2)/m);
        y2=wy1;
    }
    if(y1>wy2)
    {
        x1=x1-((y1-wy2)/m);
        y1=wy2;
    }
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    cout<<"\n\n\t\t\t\tAfter Clipping";
    rectangle(wx1,wy1,wx2,wy2);
    line(x1,y1,x2,y2);
    getch();
}
```


Output:

Enter the clip window coordinates: 200 200 400 400

Enter the line end points: 150 150 350 500



Result:

Thus the program to implementation of the line clipping has been implemented and the output was verified.

Ex. No : 5

Date :

Implementation of 3D Transformations

Aim:

To implement the following 3D Transformations:

- a. Translation
- b. Rotation
- c. Scaling

Algorithm:

1. Assign the value of edge [20] [3] as a coordinates of cube.
2. Print the menu for choosing 3D Transformation.
3. If user choose translation then get the translation factor (tx, ty, tz) and draw the translated cube.
4. If user choose rotation then print the menu for choosing rotation axis and get the rotation angle.
 - a) If user choose rotation about X axis then rotate the cube along X axis and draw the rotated cube.
 - b) If user choose rotation about Y axis then rotate the cube along Y axis and draw the rotated cube.
 - c) If user choose rotation about Z axis then rotate the cube along Z axis and draw the rotated cube.
5. If user choose scaling then get the scaling factor (sx, sy, sz) and draw the scaled cube.

Program:

```
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
#include<math.h>
int gd,gm,i,ch1,ch2;
double x1,x2,y1,y2,x,y,z,theta,temp,temp1;
void draw(double[20][3]);
void main()
{
    do
    {
        double edge[20][3]={ 100,0,0,
                               100,100,0,
                               0,100,0,
                               0,100,100,
                               0,0,100,
                               0,0,0,
                               100,0,0,
```

```

        100,0,100,
        100,100,100,
        100,100,100,
        100,100,100,
        100,100,0,
        100,100,100,
        100,100,100,
        100,100,100,
        0,100,100,
        0,100,0,
        0,0,0,
        0,0,100,
        100,0,100
    };

    clrscr();
    cout<<"Choose any one 3D Transformation : ";
    cout<<"\n\t1. Translation ";
    cout<<"\n\t2. Rotation ";
    cout<<"\n\t3. Scaling ";
    cout<<"\nEnter your choice : \t";
    cin>>ch1;
    switch(ch1)
    {
        case 1:
            cout<<"\nEnter the translation factors(tx,ty,tz) : \t";
            cin>>x>>y>>z;
            draw(edge);
            for(i=0;i<20;i++)
            {
                edge[i][0]+=x;
                edge[i][1]+=y;
                edge[i][2]+=z;
            }
            draw(edge);
            break;
        case 2:
            cout<<"\n\n\t1.Rotation about X Axis ";
            cout<<"\n\t2. Rotation about Y Axis ";
            cout<<"\n\t3. Rotation about Z Axis ";
            cout<<"\nEnter your choice : \t";
            cin>>ch2;
            cout<<"\n\nEnter the angle of rotation : \t";
            cin>>theta;
            theta=(theta*3.14)/180;
            switch(ch2)
            {

```

```

        case 1:
            draw(edge);
            for(i=0;i<20;i++)
            {
                temp=edge[i][1];
                temp1=edge[i][2];
                edge[i][1]=temp*cos(theta)-temp1*sin(theta);
                edge[i][2]=temp*sin(theta)+temp1*cos(theta);
            }
            draw(edge);
            break;
        case 2:
            draw(edge);
            for(i=0;i<20;i++)
            {
                temp=edge[i][0];
                temp1=edge[i][2];
                edge[i][0]=temp*cos(theta)+temp1*sin(theta);
                edge[i][2]=-temp*sin(theta)+temp1*cos(theta);
            }
            draw(edge);
            break;
        case 3:
            draw(edge);
            for(i=0;i<20;i++)
            {
                temp=edge[i][0];
                temp1=edge[i][1];
                edge[i][0]=temp*cos(theta)-temp1*sin(theta);
                edge[i][1]=temp*sin(theta)+temp1*cos(theta);
            }
            draw(edge);
            break;
    }
    break;
case 3:
    cout<<"\n\nEnter the scaling factors (sx,sy,sz) : \t";
    cin>>x>>y>>z;
    draw(edge);
    for(i=0;i<20;i++)
    {
        edge[i][0]*=x;
        edge[i][1]*=y;
        edge[i][2]*=z;
    }
    draw(edge);

```

```
                break;
            }
        }while(ch1<4);
    }
void draw(double edge[20][3])
{
    initgraph(&gd,&gm,"C:\\TC\\BGI");
    line(320,240,320,25);
    line(320,240,550,240);
    line(320,240,150,410);
    for(int i=0;i<19;i++)
    {
        x1=edge[i][0]+edge[i][2]*(cos(2.3562));
        y1=edge[i][1]-edge[i][2]*(sin(2.3562));
        x2=edge[i+1][0]+edge[i+1][2]*(cos(2.3562));
        y2=edge[i+1][1]-edge[i+1][2]*(sin(2.3562));
        line(x1+320,240-y1,x2+320,240-y2);
    }
    getch();
    closegraph();
}
```

Output:

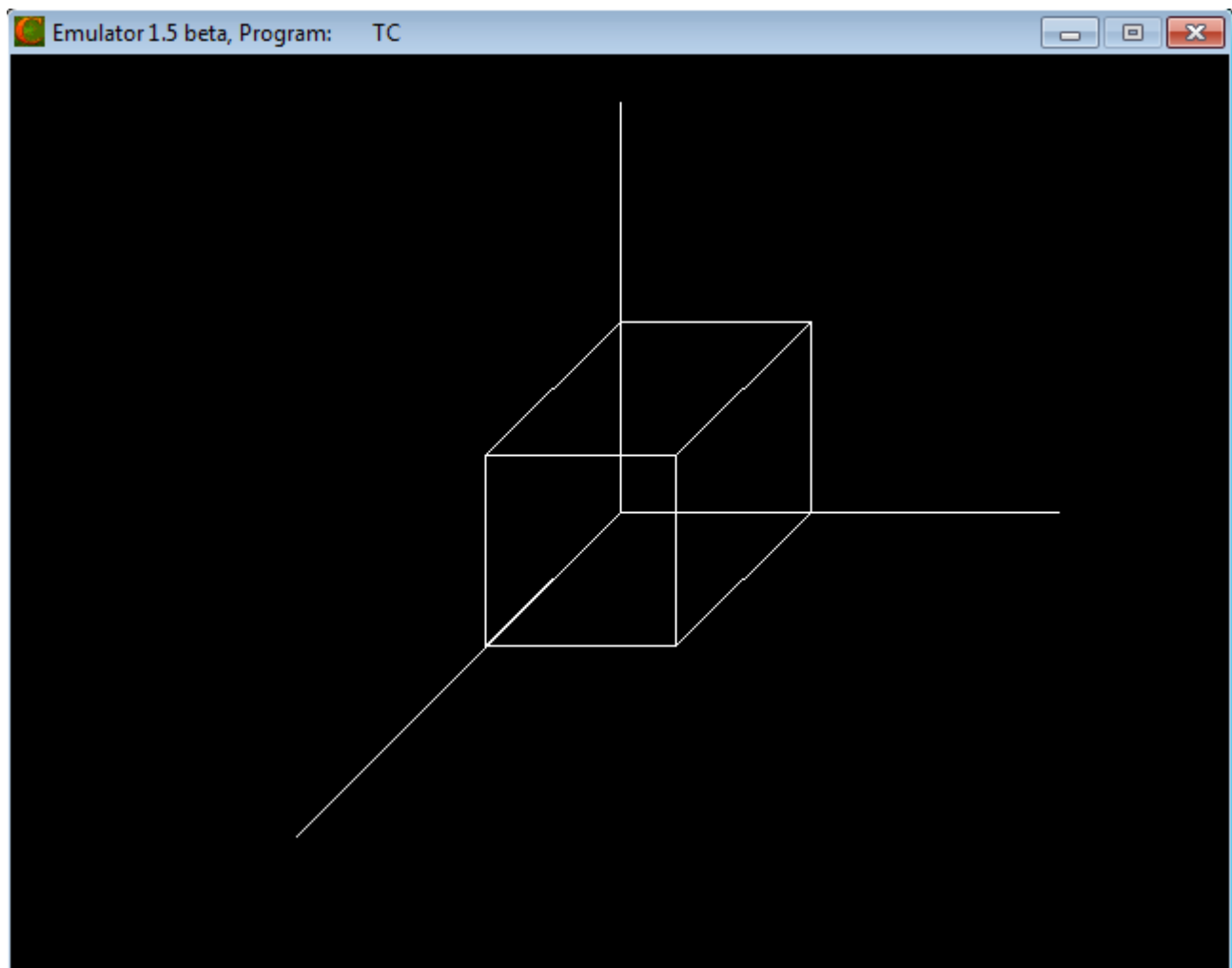
Case 1:

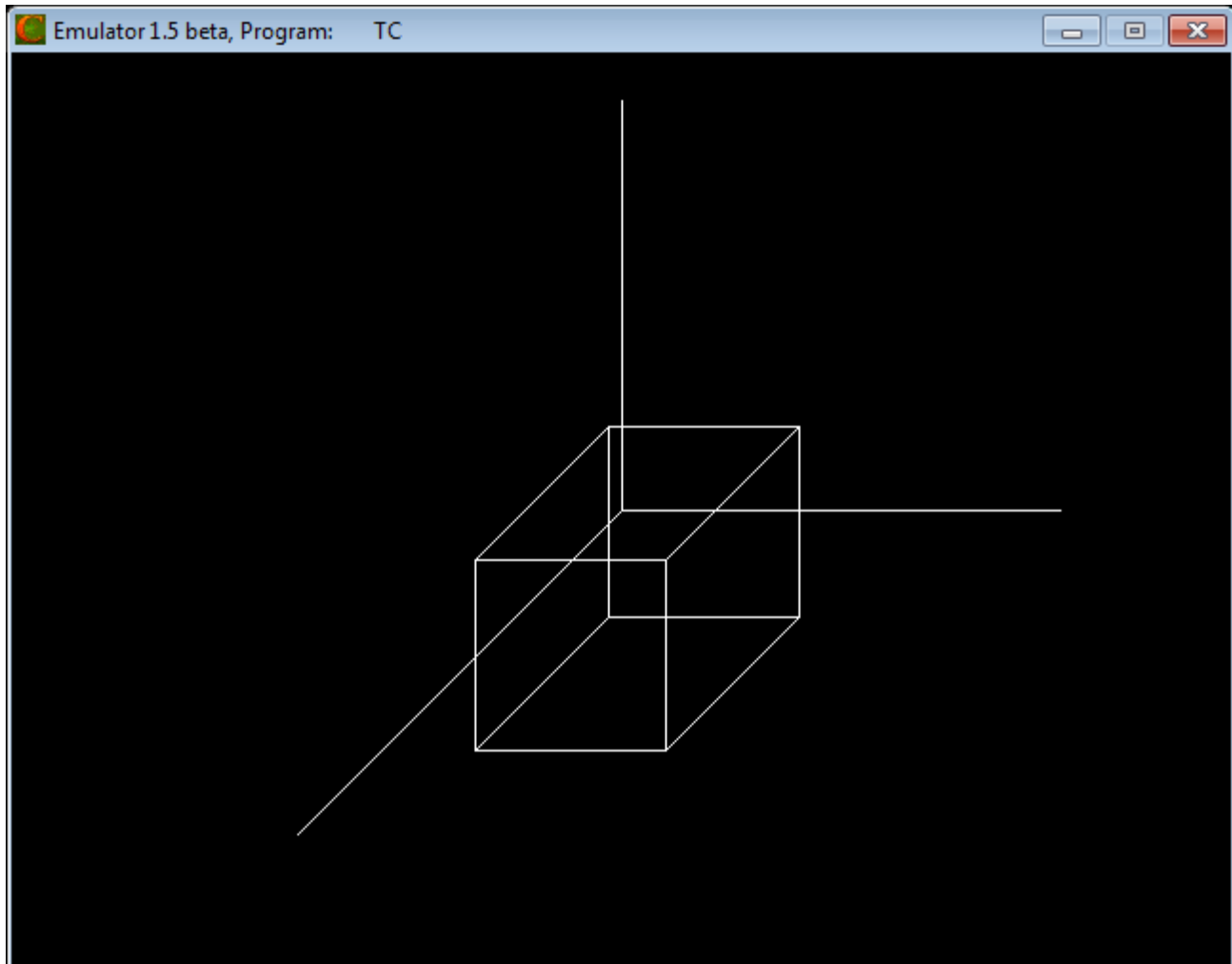
Choose any one 3D Transformation:

1. Translation
2. Rotation
3. Scaling

Enter your choice: 1

Enter the translation factors (tx,ty,tz) : 100 50 150





Case 2 (a):

Choose any one 3D Transformation:

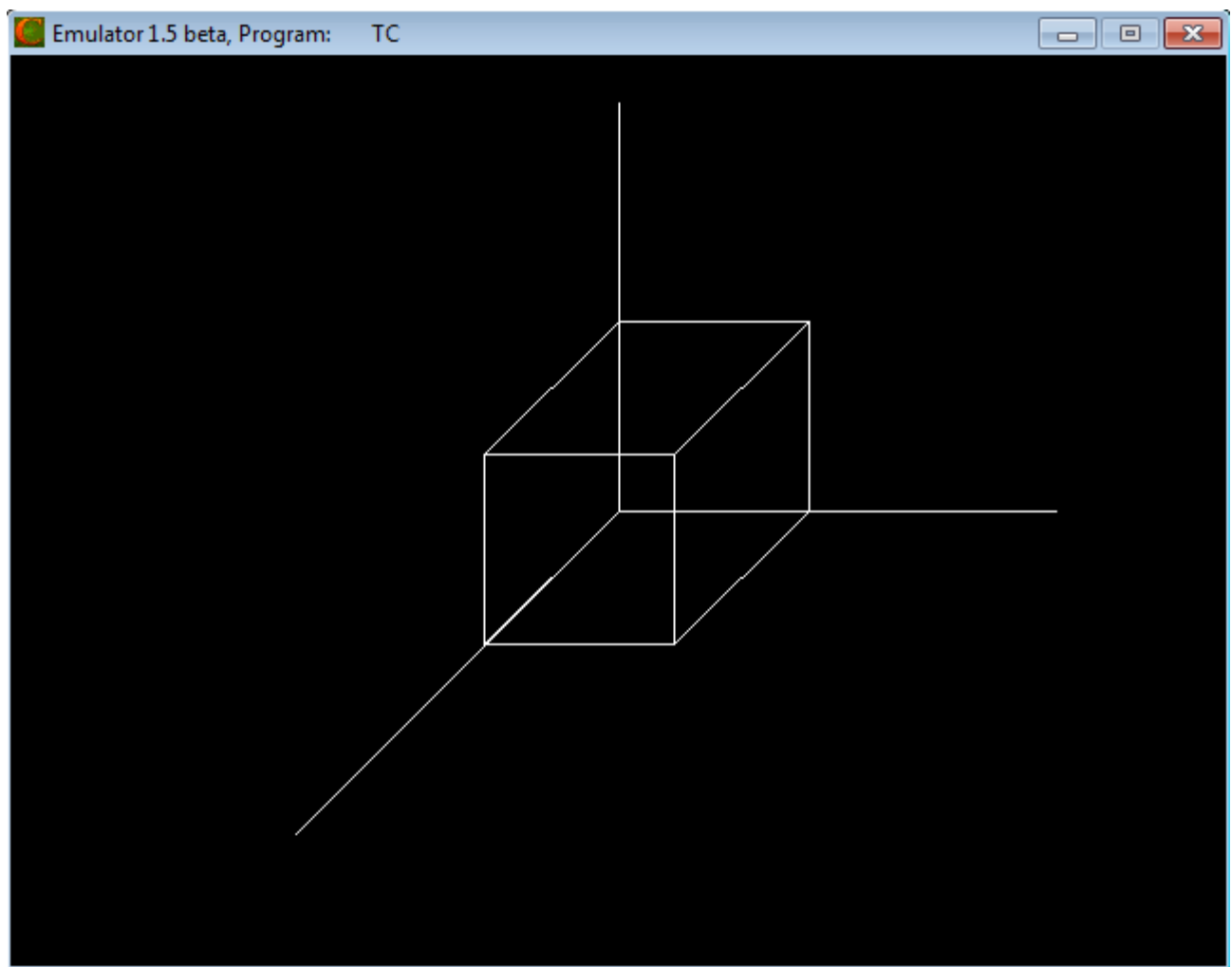
1. Translation
2. Rotation
3. Scaling

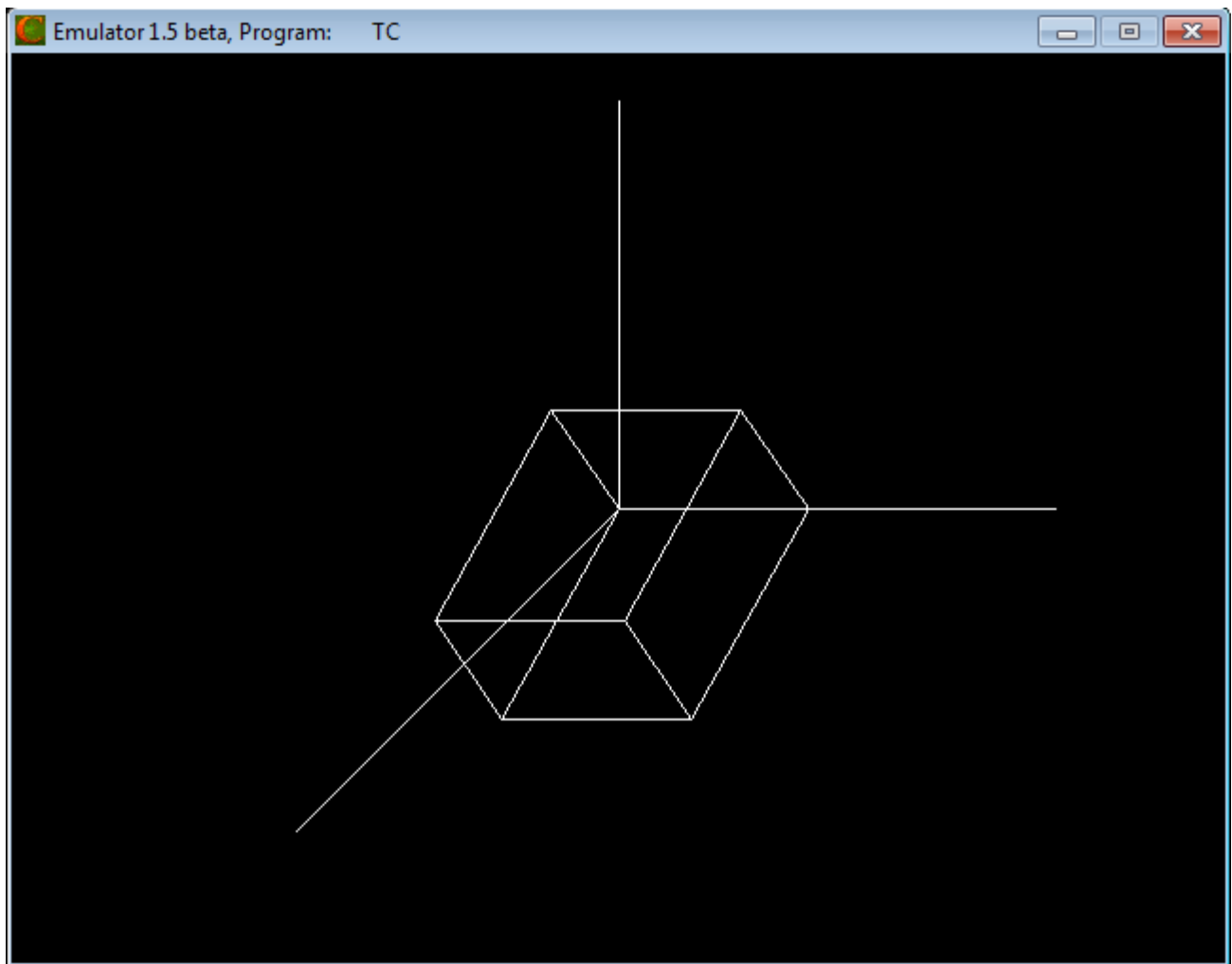
Enter your choice: 2

1. Rotation about X Axis
2. Rotation about Y Axis
3. Rotation about Z Axis

Enter your choice: 1

Enter the angle of rotation: 30





Case 2 (b):

Choose any one 3D Transformation:

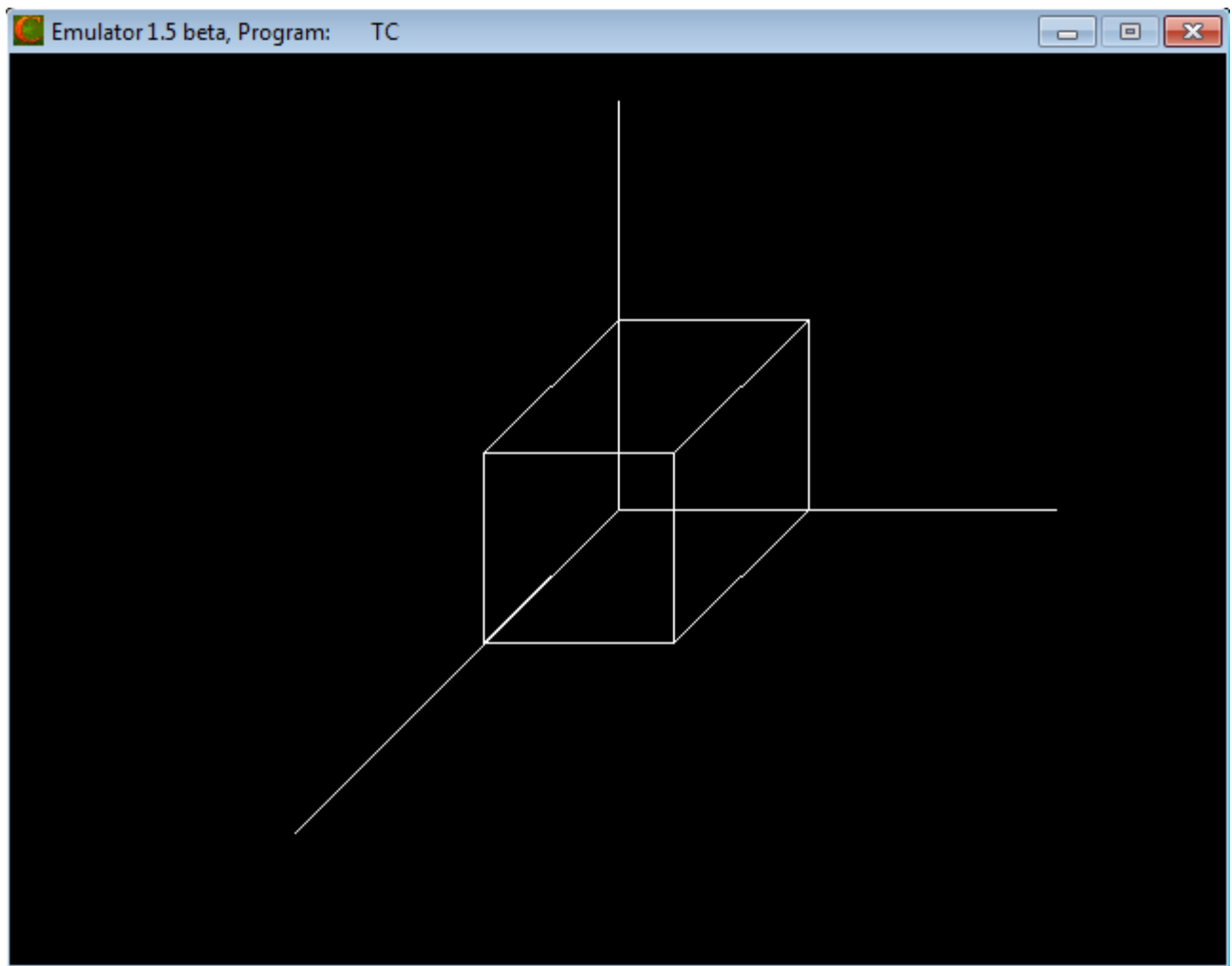
1. Translation
2. Rotation
3. Scaling

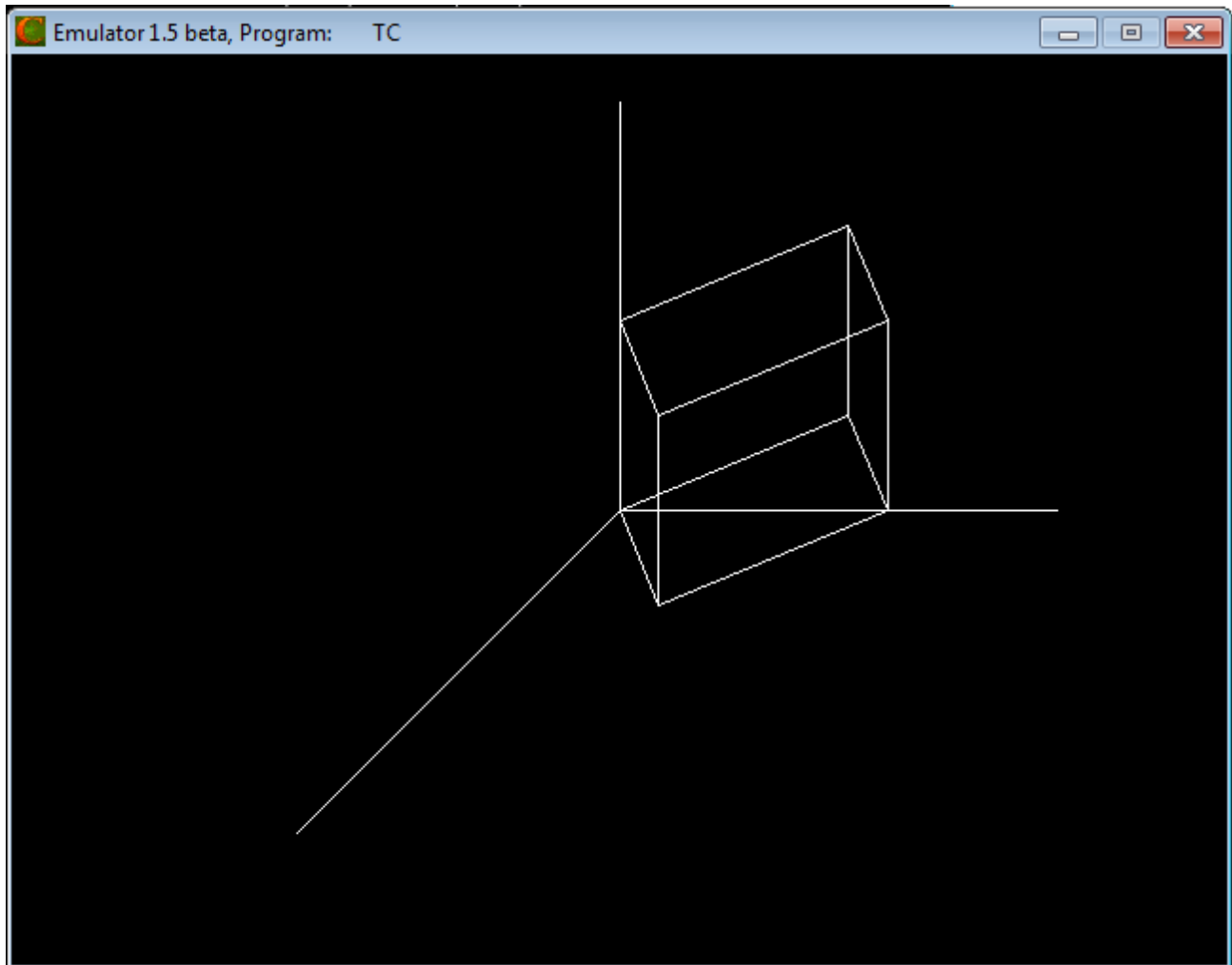
Enter your choice: 2

1. Rotation about X Axis
2. Rotation about Y Axis
3. Rotation about Z Axis

Enter your choice: 2

Enter the angle of rotation: 45





Case 2 (c):

Choose any one 3D Transformation:

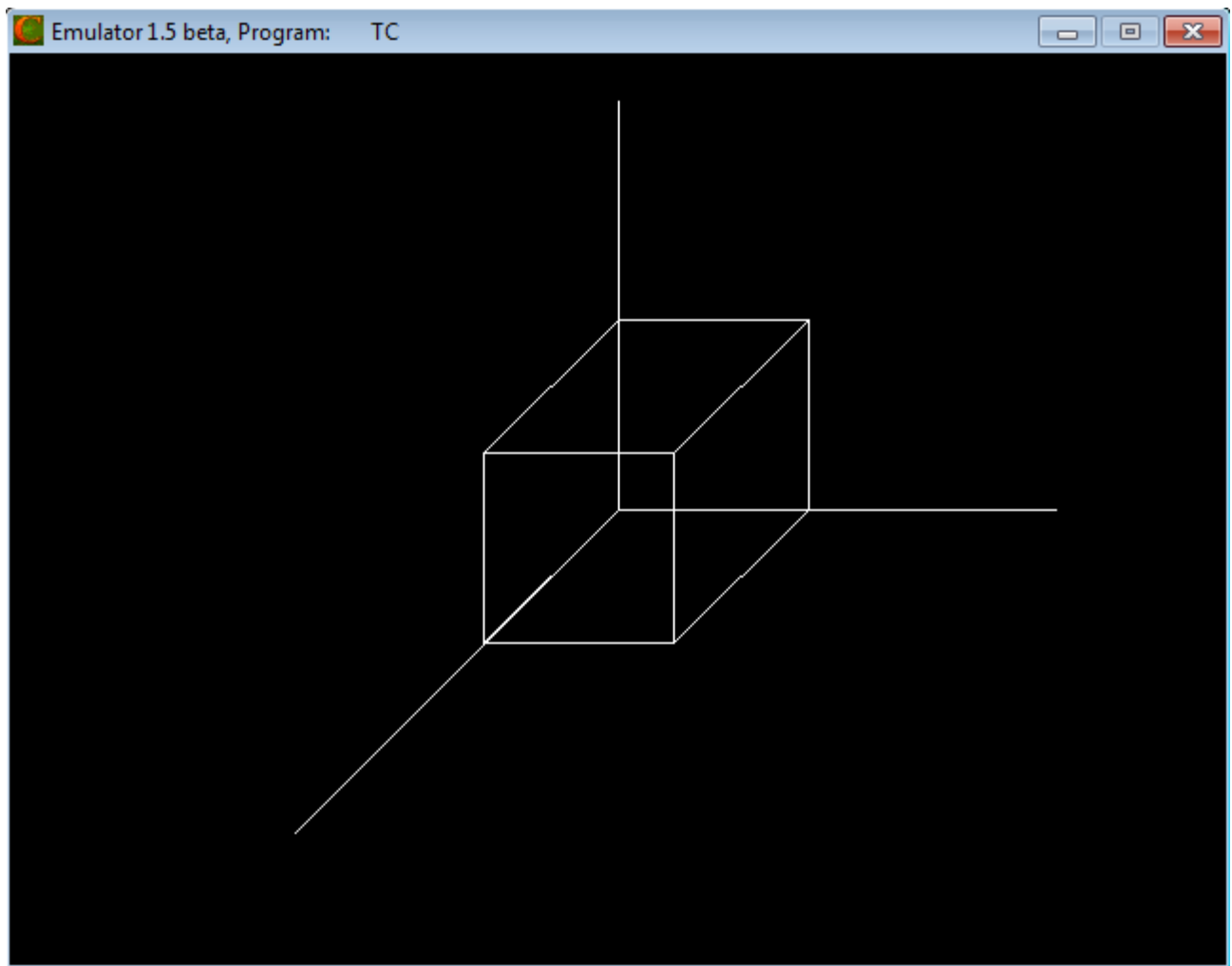
1. Translation
2. Rotation
3. Scaling

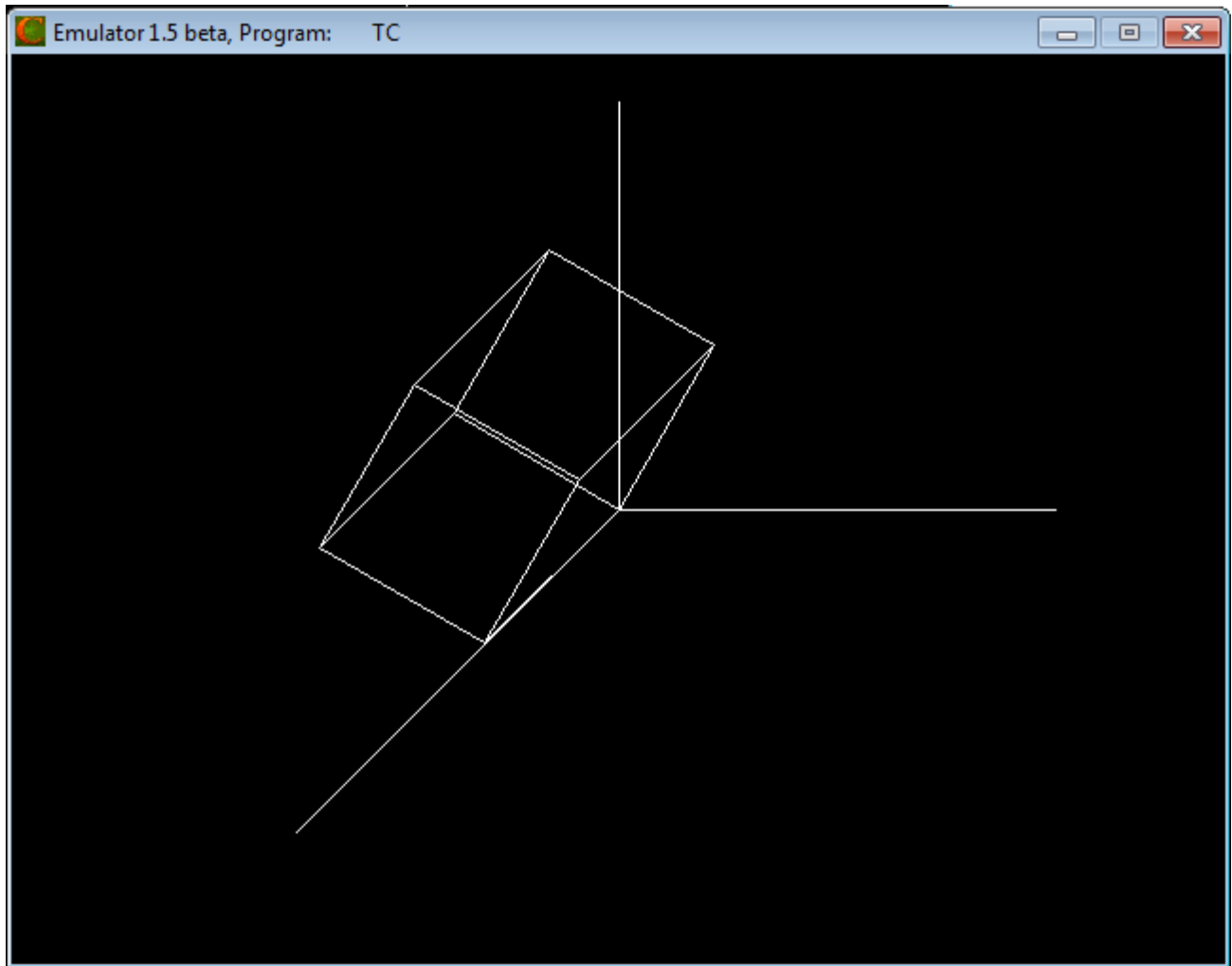
Enter your choice: 2

1. Rotation about X Axis
2. Rotation about Y Axis
3. Rotation about Z Axis

Enter your choice: 3

Enter the angle of rotation: 60





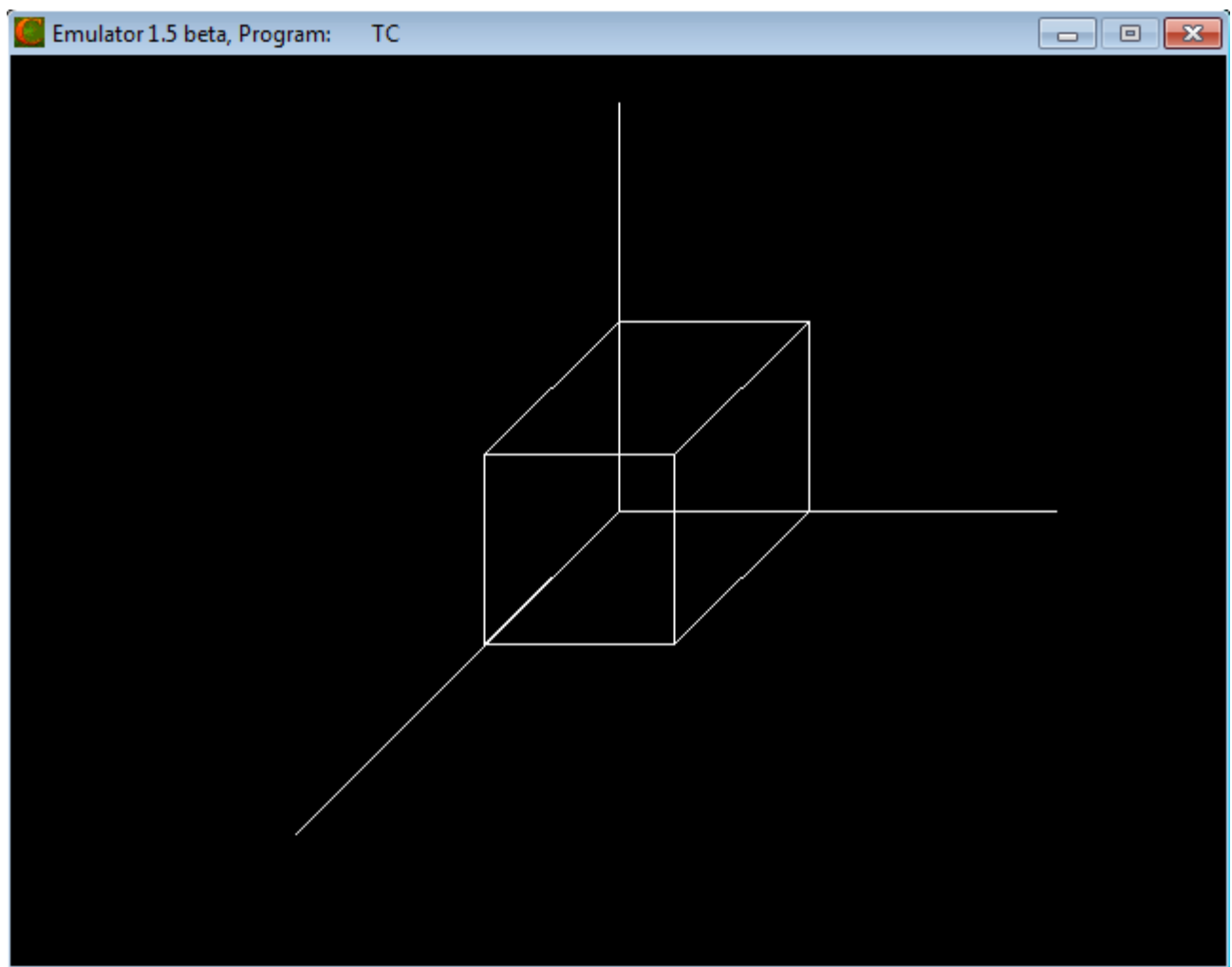
Case 3:

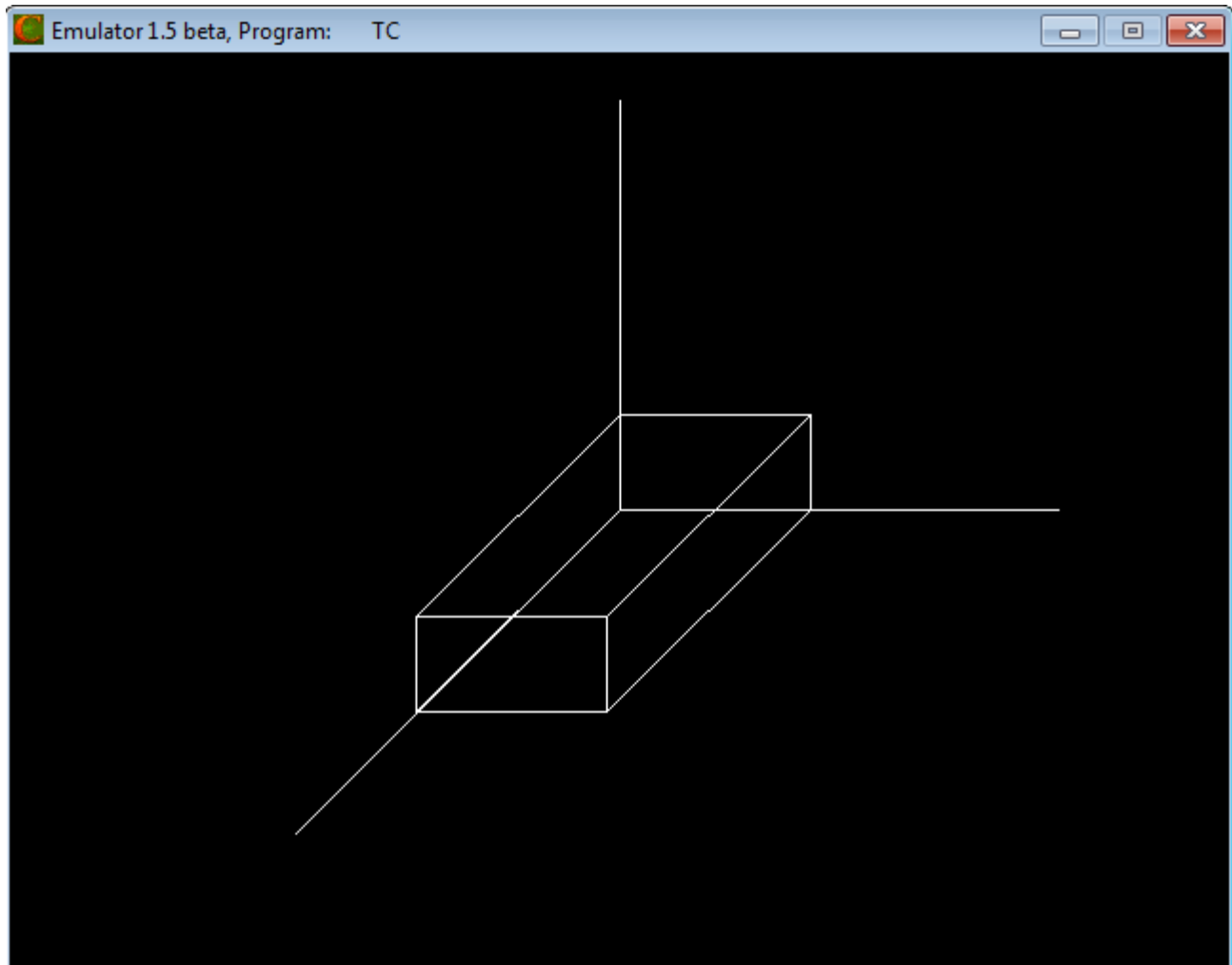
Choose any one 3D Transformation:

1. Translation
2. Rotation
3. Scaling

Enter your choice: 3

Enter the scaling factors (sx,sy,sz) : 1 0.5 1.5





Result:

Thus the implementation of the 3D Transformations (translation, rotation and scaling) has been implemented and the output was verified

Ex. No : 6

Date :

3D Projections - Parallel and Perspective

Aim:

To write a program to implement the following 3D projections:

- a. Parallel Projection
- b. Perspective Projection

Algorithm:

1. Get the coordinate to draw the cube.
2. Print the menu : 1. Parallel Projection 2. Perspective Projection
3. If user choose Parallel Projection then find the coordinates of following views and draw them:
 - a) Side View
 - b) Front View
 - c) Top View
4. If user choose Perspective Projection then simply draw the cube using bar3d() function.

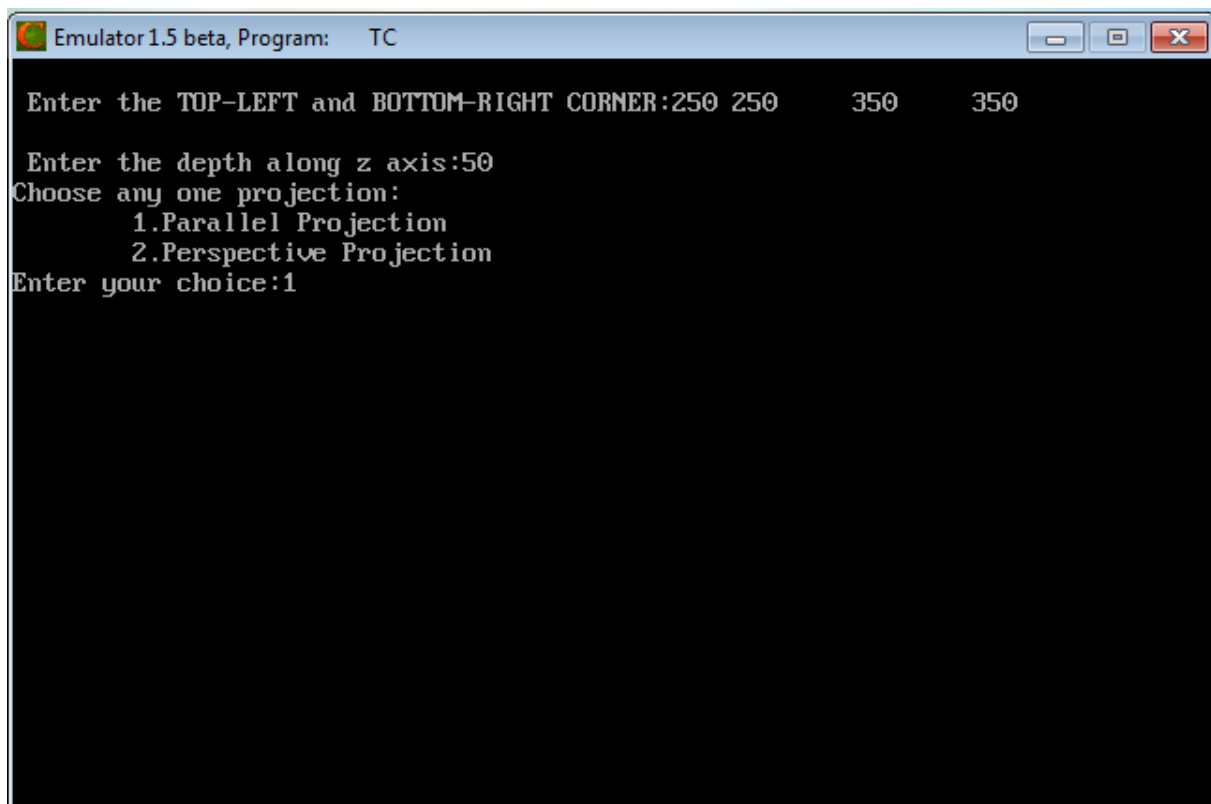
Program:

```
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
int gd,gm,x1,y1,x2,y2,dep,ch;
void main()
{
    cout<<"\n Enter the TOP-LEFT and BOTTOM-RIGHT CORNER:";
    cin>>x1>>y1>>x2>>y2;
    cout<<"\n Enter the depth along z axis:";
    cin>>dep;
    do
    {
        cout<<"Choose any one projection:\n\t1.Parallel Projection\n\t2.Perspective
Projection\nEnter your choice:";
        cin>>ch;
        initgraph(&gd,&gm,"C:\\\\TC\\\\BGI");
        switch(ch)
        {
            case 1:
                rectangle(x2+100,y1,x2+100+dep,y2);
                outtextxy(x2+100,y1-10,"SIDE VIEW");
                rectangle(x1,y1,x2,y2);
                outtextxy(x1,y1-10,"FRONT VIEW");
                rectangle(x1,y1-(y2-y1),x2,x1+dep-(y2-y1));
                outtextxy(x1,y1-(y2-y1)-10,"TOP VIEW");
                getch();
                closegraph();
```



```
                break;
            case 2:
                bar3d(x1,y1,x2,y2,dep,1);
                getch();
                closegraph();
                break;
        }
    }while(ch<3);
}
```

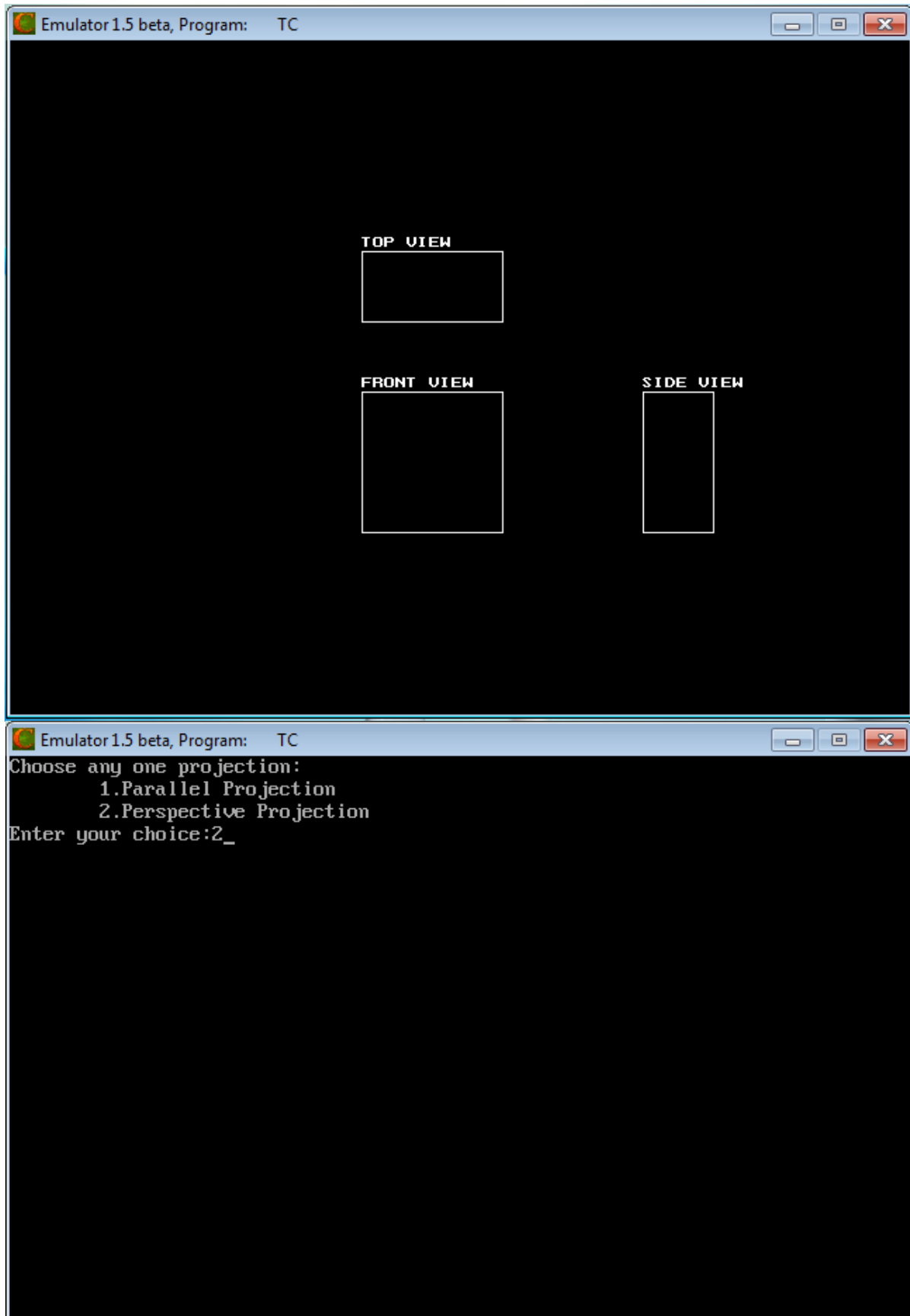
Output:

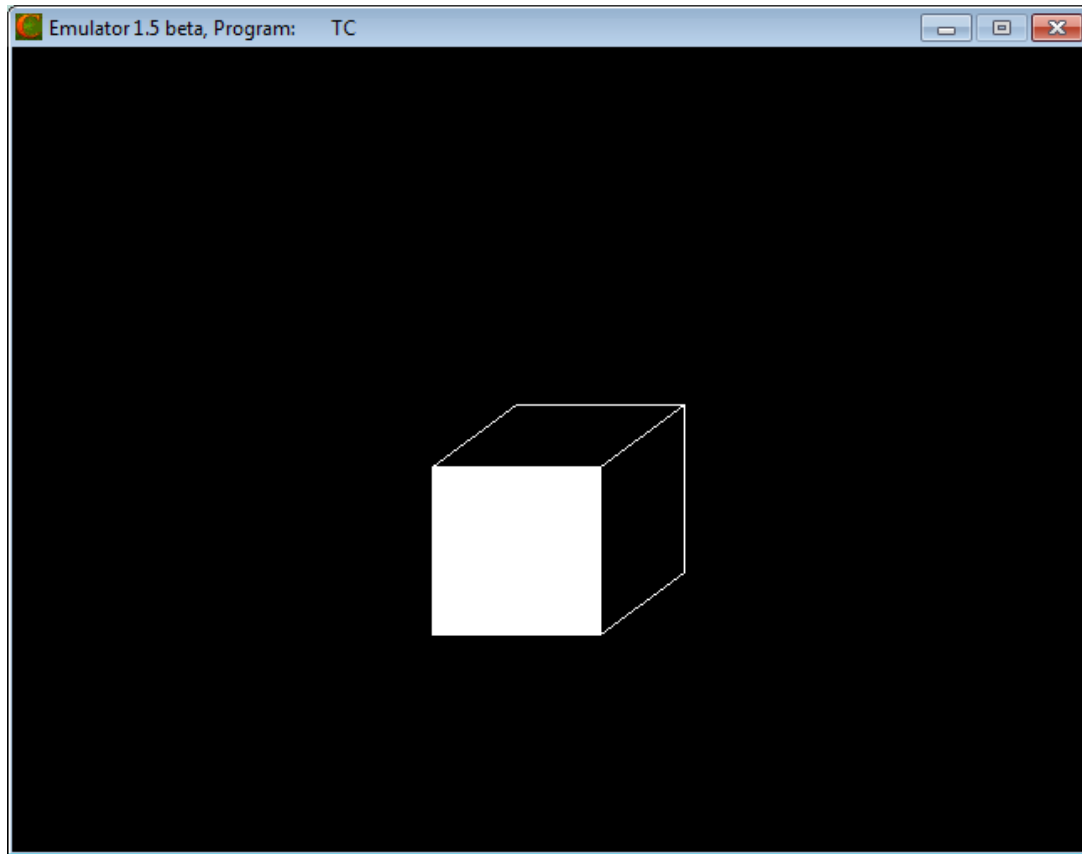


The screenshot shows a window titled "Emulator 1.5 beta, Program: TC". The window contains the following text:

```
Enter the TOP-LEFT and BOTTOM-RIGHT CORNER:250 250      350      350

Enter the depth along z axis:50
Choose any one projection:
    1.Parallel Projection
    2.Perspective Projection
Enter your choice:1
```





Result:

Thus the implementations of 3D projections (Parallel and Perspective) has been created and the output was verified.

Ex. No : 7

Date :

Creating 3D Scenes

Aim:

To create the 3D Scene in C++.

Algorithm:

1. Set the background color.
2. Find the coordinates to draw a 3D scene.
3. Plot that coordinates using pre-defined functions like line(), rectangle(), fillellipse().
4. Set the color of the objects by setcolor().

Program:

```
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
intgd, gm;
void main()
{
    initgraph(&gd, &gm, "C:\\TC\\BGI");
    setbkcolor(BLUE);
    setcolor(RED);
    rectangle(150, 150, 250, 200);
    line(150, 200, 120, 180);
    line(120, 180, 120, 130);
    setcolor(LIGHTRED);
    line(120, 130, 150, 150);
    line(150, 150, 200, 100);
    line(200, 100, 250, 150);
    fillellipse(200, 125, 10, 10);
    line(120, 130, 170, 80);
    line(170, 80, 200, 100);
    setcolor(BROWN);
    rectangle(190, 170, 210, 200);
    line(190, 170, 200, 180);
    line(200, 180, 200, 200);
    setcolor(CYAN);
    line(190, 200, 190, 270);
    line(210, 200, 210, 250);
    line(190, 270, 300, 270);
    line(210, 250, 300, 250);
    setcolor(LIGHTGRAY);
```

```
circle(230,260,5);  
circle(270,260,5);  
setcolor(BROWN);  
line(230,259,270,259);  
line(230,261,270,261);  
line(230,230,230,261);  
line(232,230,232,260);  
line(227,230,237,230);  
line(227,231,237,231);  
line(270,260,250,245);  
line(271,260,251,245);  
line(250,245,230,245);  
line(250,246,230,246);  
line(248,245,248,240);  
line(247,245,247,240);  
fillellipse(248,238,3,1);  
setcolor(LIGHTGRAY);  
line(252,245,265,245);  
line(252,246,265,246);  
getch();  
}
```

Output:



Result:

Thus the 3D scene has been created and the output was verified.

Ex. No : 8

Date :

Image Editing and Manipulation

Aim:

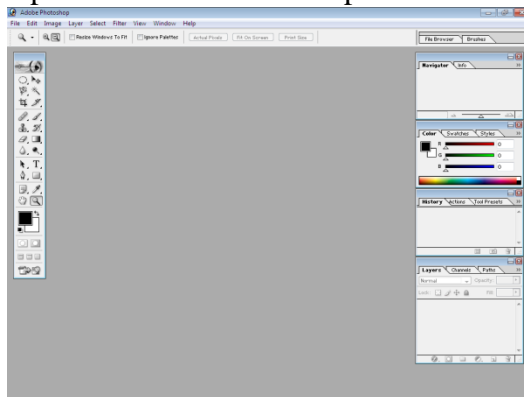
To perform the following operations,

- (i) Basic Operations on image using Adobe Photoshop 7.0
- (ii) Creating gif animated images in PhotoScape
- (iii) To optimize the image in Adobe Photoshop 7.0

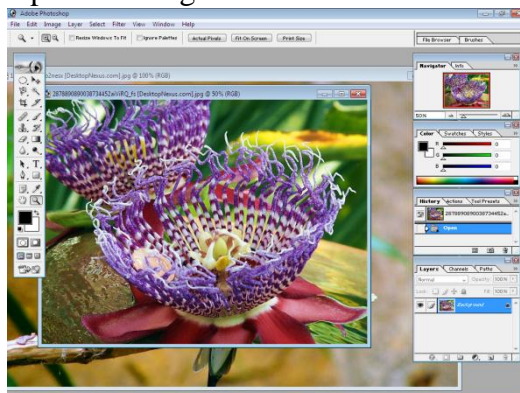
Procedure:

1. To perform the basic operations on image using Adobe Photoshop 7.0

- i. Open the Adobe Photoshop 7.0.



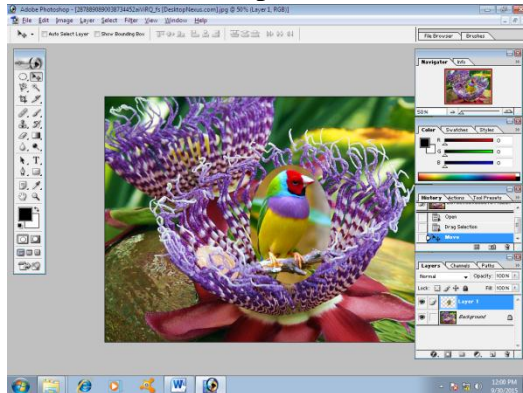
- ii. Open the images.



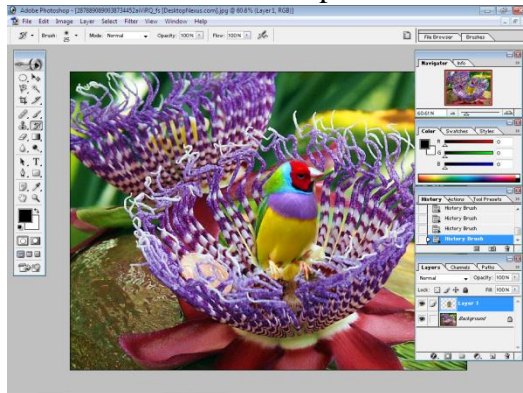
- iii. Select the particular portion of the image by Elliptical Marquee Tool.



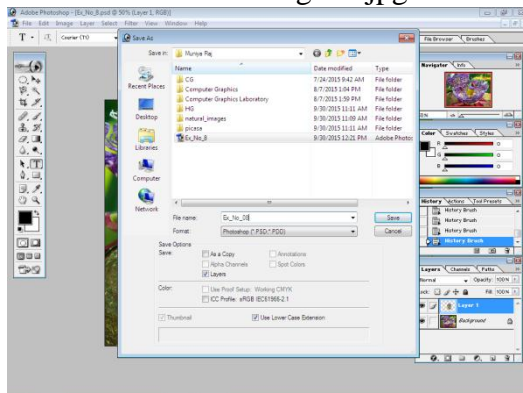
- iv. Move that selected portion to another image by using Move Tool.



- v. Eliminate the unwanted portion of the image by using History Brush Tool.

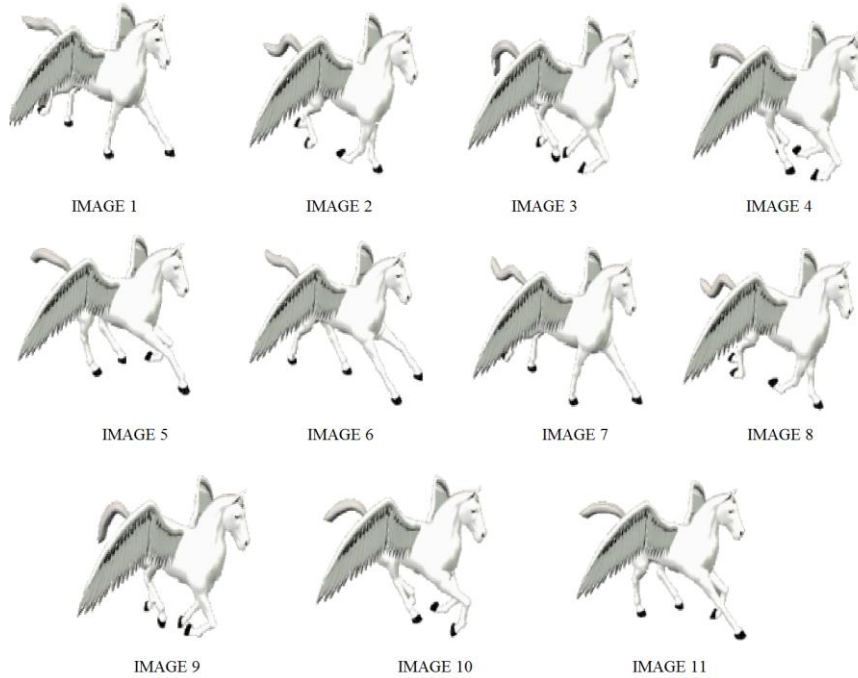


- vi. Save the modified image in jpg format

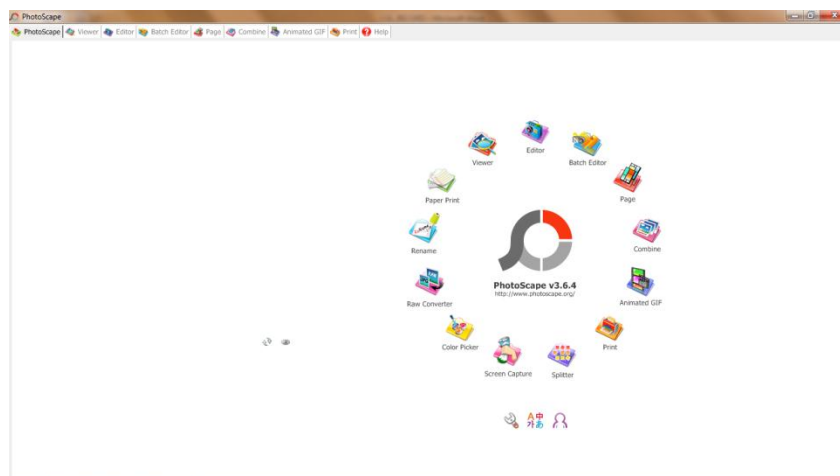


2. To Create the gif animated images in PhotoScape

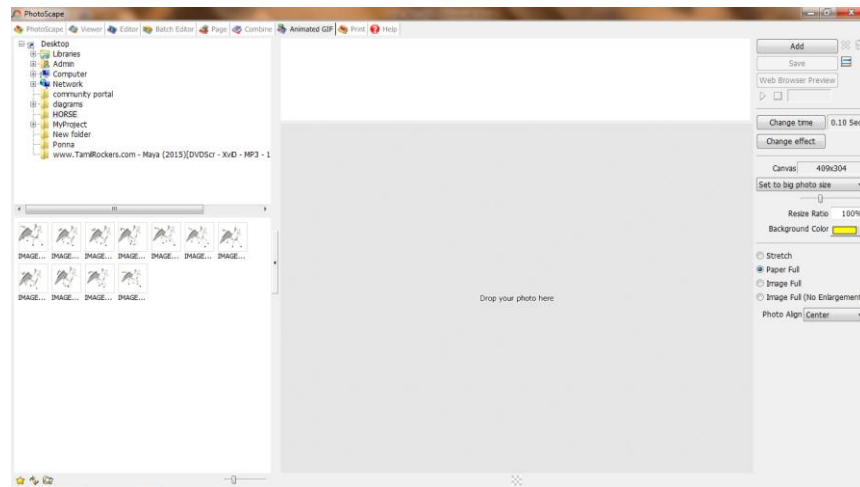
- i. Draw the images in different scenes to animate. Save all the images in jpg format.



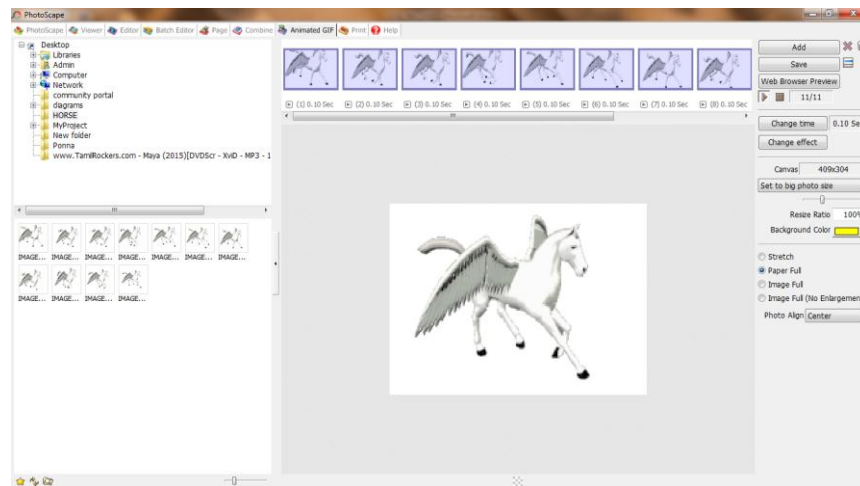
- ii. Open PhotoScape.



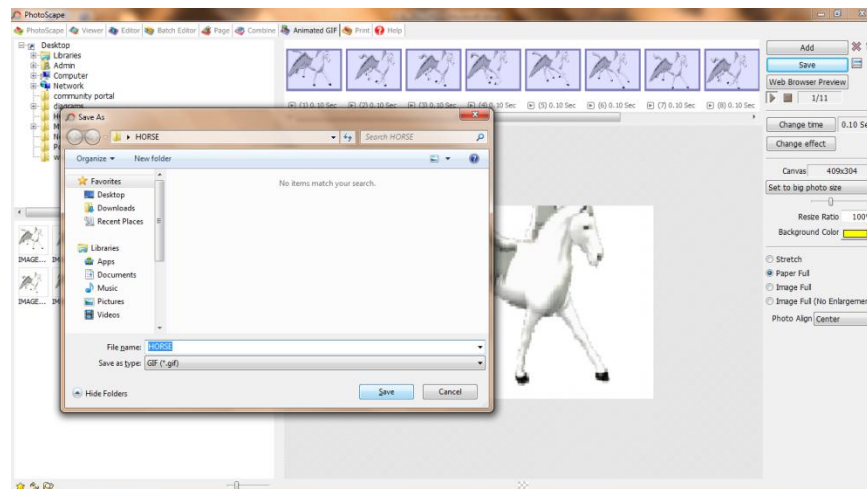
iii. Choose Animated GIF from the menu.



iv. Drop all the images in center bottom portion of the window.

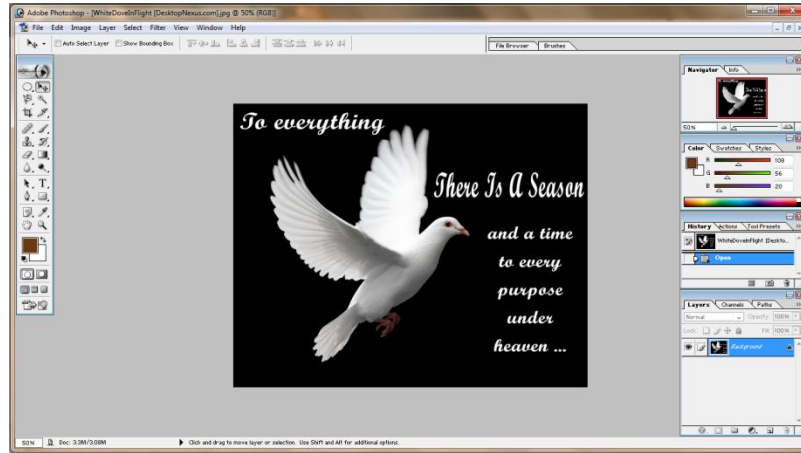


v. Save the animated file in gif format.



3. To optimize the image in Adobe Photoshop 7.0

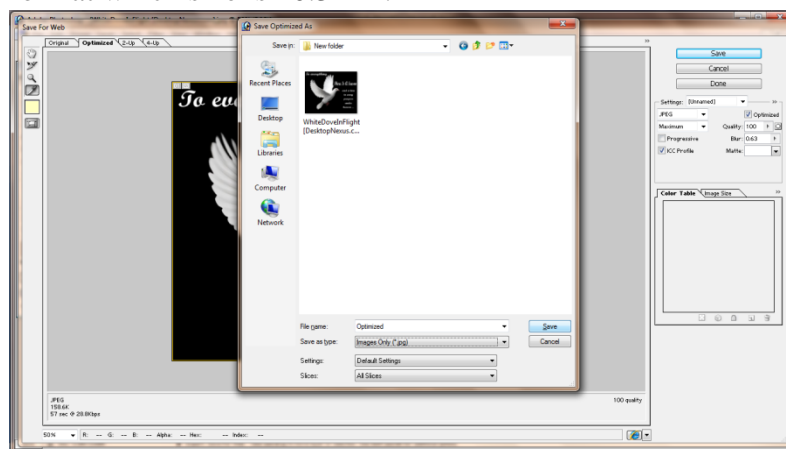
- i. Open the image in Adobe Photoshop 7.0 which size is 85.2 KB



- ii. File • Save for Web (or press Alt + Shift + Ctrl + S)



- iii. Adjust the Quality and Blur of the image. Finally the optimized image in jpg format which size is 40.3 KB.



Result:

Thus the operations (Basic Operations on image, creating gif animated image, optimize the image) has been performed and the output was verified.

Ex. No : 9

Date :

2D Animation

Aim:

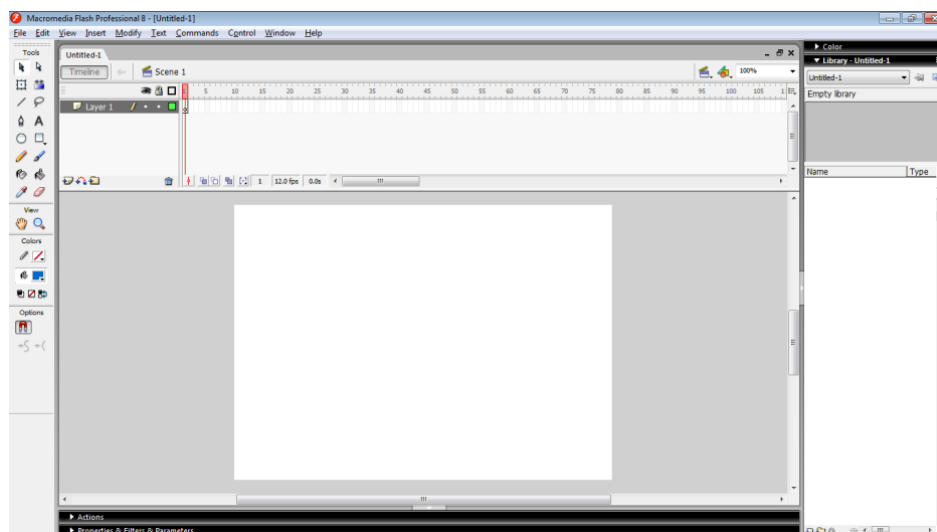
To create interactive animation using Macromedia Flash 8.

Procedure:

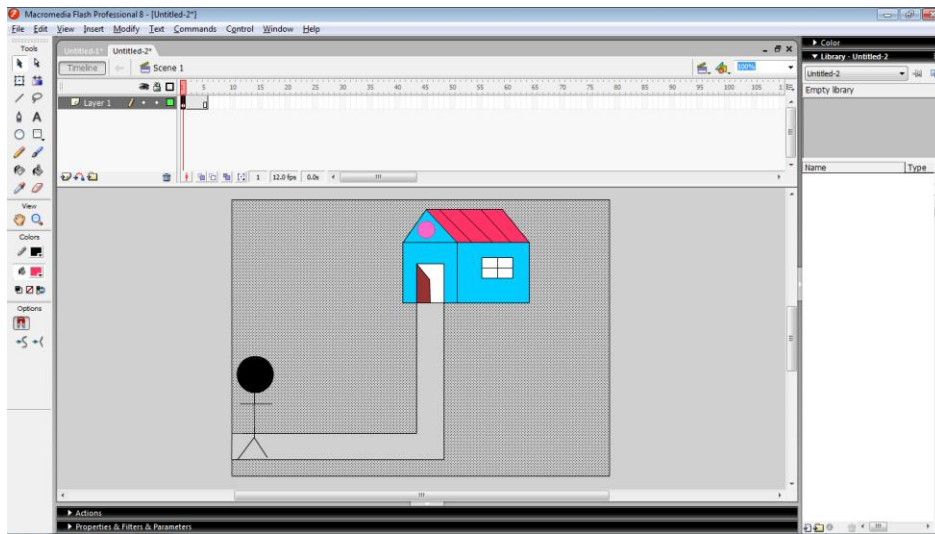
1. Open Macromedia Flash 8.



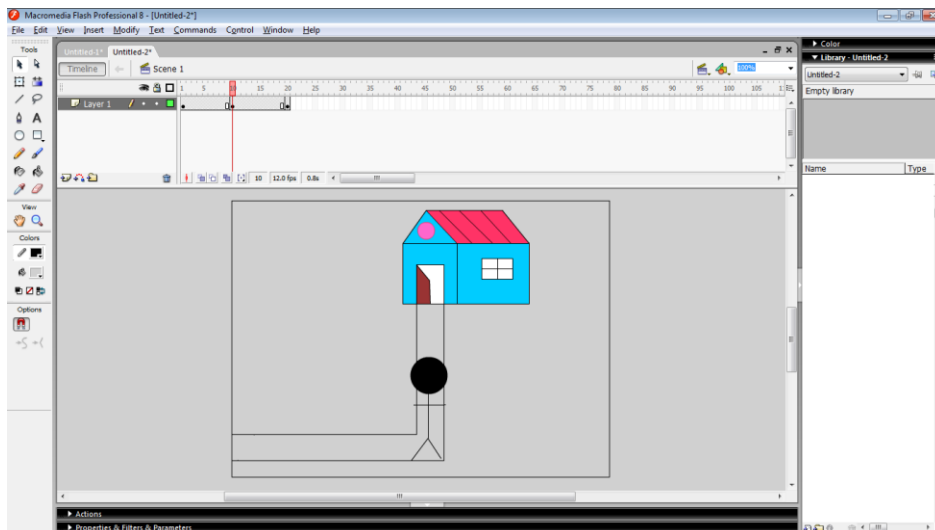
2. Create New • Flash Document.



3. Draw the 2D image (scene) to animate.



4. Insert a new frame and change the scene for new frame.



5. Similarly create many frames.
6. Control • Play (or press Enter) to run the Animation.

Result:

Thus the interactive 2D Animation has been created and the output was verified.