DDA ALGORITHM

EX NO: 1

Aim :

To write a C program to draw a line using DDA Algorithm.

Algorithm:

Step 1: Start the program.

Step 1: Input the line endpoints and store the left endpoint in (x1, y1) and right endpoint in (x2, y2)

Step 2: Calculate the values of Δx and Δy using

Δx = xb – xa, Δy = yb – ya

Step 3: if the values of Δx > Δy assign values of steps as Δx otherwise the values of steps as Δy

Step 4: Calculate the values of X increment and Y increment and assign

the value x= xa and y = ya.

Step 5: for k=1 to steps do **X = X + X increment** , **Y= Y + Y increment**

Putpixel(ceil(x), ceil(y),15).

Step 6: Stop the Program.

PROGRAM :

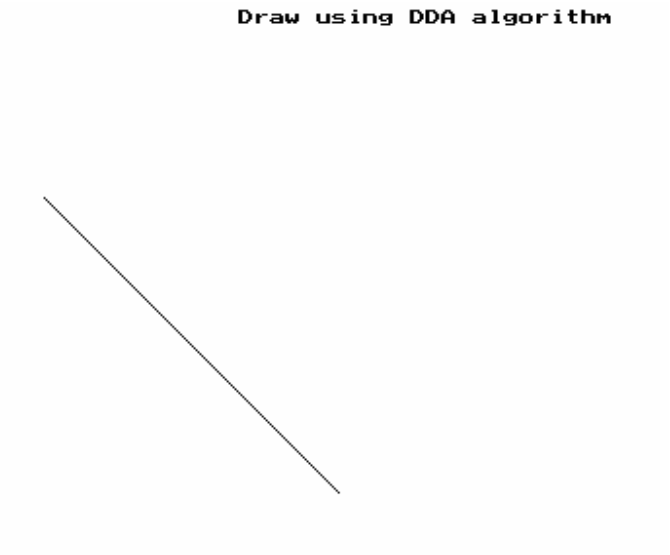
```
#include<stdio.h>

#include<conio.h>

#include<stdlib.h>

#include<graphics.h>

void main()

{

        int i,steps,x1,x2,y1,y2;

        float x,y,xinc,yinc,dx,dy;

        char msg[86];

        int gdriver = DETECT,gmode,errorcode;

        clrscr();

        initgraph(&gdriver,&gmode,"f:\\tc");

        printf("\n Enter the co ordinates ");

        scanf("%d%d%d%d",&x1,&x2,&y1,&y2);

        cleardevice();

        outtextxy(200,4,"Draw Using DDA");

        line(x1,x2,y1,y2);

        dx = x2 - x1;

        dy = y2 - y1;

        if(abs(dx) > abs(dy))
```

```
            steps = abs(dx);
        else
            steps = abs(dy);
        xinc = (float)dx/steps ;
        yinc = (float)dy/steps ;
        y = y1;
        x = x1;
        putpixel(ceil(x),ceil(y),20);
        for(i = 0;i <= steps ;i++)
        {
            x += xinc ;
            y += yinc ;
            putpixel(x,y,2);
            delay(45);
        }
        getch();
}
```

OUTPUT :

Enter the Co-ordinates :

50 100 200 300



RESULT:

Thus the above program has been executed and output is verified.

BRESENHAM'S LINE DRAWING ALGORITHM

EX NO: 2

Aim :

To write a C program to draw a line using Bresenham's Algorithm.

Algorithm:

Step 1: Start the program.

Step 2: Input the two endpoints (x1,y1) and (x2,y2).

Step 3: Plot the pixel value (x1,y1) with a specified color.

Step 4: Calculate the value of dx and dy and find the starting value of decision parameter as **dp=2*dy-dx.**

Step 5: Calculate the values of s1 and s2 depending on (x1,y1) and (x2,y2) values.

Step 6: If dp<0, the next point to plot is (x,y+s2) and dp=+2*dy.

Step 7: Otherwise the next point to plot is (x+s1,y+s2) and

dp=dp+2*dx-2*dy.

Step 8: Repeat steps 5 and 6 dx times.

Step 9: Stop the program.

PROGRAM :

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

void main()
{
        int x1,x2,y1,y2;
        int gd=DETECT,gm;
        void linebres(int,int,int,int);
        printf("Enter the two end points:");
        scanf("%d%d%d%d",&x1,&x2,&y1,&y2);
        initgraph(&gd,&gm,"");
        cleardevice();
        linebres(x1,y1,x2,y2);
        getch();
        line(x1,y1,x2,y2);
        getch();
        closegraph();
}
void linebres(int x1,int y1,int x2,int y2)
{
        int dx=abs(x1-x2),dy=abs(y1-y2);
        int p,x,y,i,xend,yend;
```

```
if(dx!=0)
{
        p=2*dy-dx;
        if(x1>x2)
        {
                x=x2;
                y=y2;
                xend=x1;
        }
        else
        {
                x=x1;
                y=y1;
                xend=x2;
        }
        putpixel(x,y,2);
        for(i=x;i<xend;i++)
        {
                x+=1;
                if(p<0)
                p+=2*dy;
                else
                p+=2*(dy-dx);
        }
                putpixel(x,y,2);
```

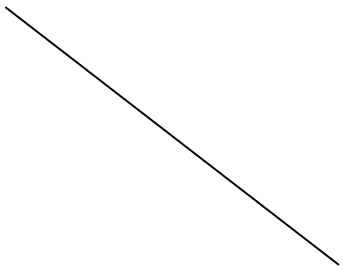
```
        }
        else
        {
                p=2*dx-dy;
                if(y1>y2)
                {
                        x=x2;
                        y=y2;
                        yend=y2;
                }
                putpixel(x,y,2);
                for(i=y;i<yend;i++)
                {
                        y+=1;
                        if(p<0)
                        p+=2*dx;
                        else
                        {
                                x+=1;
                                p+=2*(dx-dy);
                        }
                        putpixel(x,y,2);
                }
        }
}
```

OUTPUT :

Bresenham line drawing algorithm

Enter the co-ordinates

150 100 250 300



RESULT:

Thus the above program has been executed and output is verified.

BRESENHAM'S CIRCLE DRAWING ALGORITHM

EX NO: 3

Aim :

To write a C program to draw a Circle using Bresenham's Algorithm.

Algorithm:

Step 1: Start the program.

Step 2: Input radius r and the midpoint of the circle (x,y) and obtain the first point on the circumference for the circle as (0,r).

Step 3: Calculate the initial value of the decision parameter as p=1-r.

Step 4: At each position check the following conditions.
a) If p<0 then x=x+1 and p+=2*x+1
b) Else y=y-1 and p+=2*(x-y)+1.

Step 5: Determine symmetry points at the other seven octants.

Step 6: Move each calculated pixel position (x,y) onto the circular path centered on (xc,yc) and plot the coordinate value as x=x+xc and y=y+yc.

Step 7: Repeat step 3 until x<y.

Step 8: Stop the program.

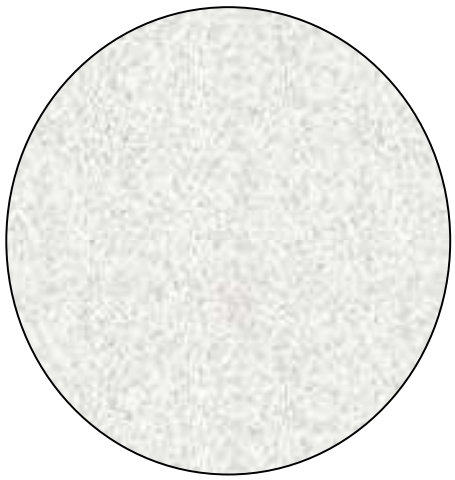
PROGRAM :

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
void main()
{
   int gd=DETECT,gm;
   int x,y,r;
   void cir(int,int,int);
   printf("Enter the Mid points and Radious:");
   scanf("%d%d%d",&x,&y,&r);
   initgraph(&gd,&gm,"");
   cir(x,y,r);
   getch();
   closegraph();
}
void cir(int x1,int y1,int r)
{
   int x=0,y=r,p=1-r;
   void cliplot(int,int,int,int);
   cliplot(x1,y1,x,y);
   while(x<y)
  {
      x++;
      if(p<0)
```

```
        p+=2*x+1;
      else
      {
        y--;
        p+=2*(x-y)+1;
      }
      cliplot(x1,y1,x,y);
   }
 }
void cliplot(int xctr,int yctr,int x,int y)
{
   putpixel(xctr +x,yctr +y,1);
   putpixel(xctr -x,yctr +y,1);
   putpixel(xctr +x,yctr -y,1);
   putpixel(xctr -x,yctr -y,1);
   putpixel(xctr +y,yctr +x,1);
   putpixel(xctr -y,yctr +x,1);
   putpixel(xctr +y,yctr -x,1);
   putpixel(xctr -y,yctr -x,1);
   getch();
}
```

OUTPUT:

Enter the Mid points and Radious:100 100 50



RESULT:

Thus the above program has been executed and output is verified.

BRESENHAM'S ELLIPSE DRAWING ALGORITHM

EX NO: 4

Aim :

To write a C program to draw a Ellipse using Bresenham's Algorithm

Algorithm:

Step 1: Start the program.

Step 2: Input r_x , r_y and the center of the ellipse (x_c,y_c) and obtain the first point on the ellipse centered on the origin as $(x_0,y_0) = (0,r_y)$.

Step 3: Calculate the initial value of the decision parameter in region1 as $P1_0 = r_y^2 - r_x^2 r_y + \frac{1}{4} r_x^2$

Step 4: At each position *k x* in region 1 , starting at k=0,perform the following test. If $p1_k < 0$ the next point along the ellipse centered on (0,0) is (x_{k+1},y_k) and $p1_{k+1} = p1_k + 2r_y^2 xk+1 + r_y^2$

Step 5: Otherwise the next point along the ellipse is (x_k+1 ,y_k-1) and

$p1_{k+1} = p1_k + 2r_y^2 x_{k+1} - 2 r_x^2 y_{k+1} + r_y^2$ with

$2r_y^2 x_{k+1} = 2r_y^2 x_k + 2r_y^2$, $2r_x^2 y_{k+1} = 2r_x^2 y_k - 2r_x^2$ and continue until $2r_y^2 x \geq 2r_x^2 y$.

Step 6: Calculate the initial position of the decision parameter in region 2 as

$P20 = ry2 \left(x0 + \frac{1}{2}\right)^2 + r_y^2 (y_{0-1})^2 - r_x^2 r_y^2$ where (x_0 ,y_0) is the last position calculated in region 1.

Step 7: At each y_k position in region 2, starting at k=0,perform the following test , if $p2_k > 0$ the next point along the ellipse centered on (0,0) is (x_k , y_{k+1}) and $p2_{k+1} = p2_k - 2r_x^2 y_{k+1} + r_y^2$.

Step 8: Otherwise the next point along the ellipse is $(x_k + 1 ,y_k -1)$ and

$P2_{k+1} = p2_k - 2r_y^2 x_{k+1} - 2r_x^2 y_{k+1} + r_x^2$.

Step 9: Using the same incremental values for x and y as in region 1 continue until y=0.

Step 10: For both regions determine symmetry points along the other three quadrants.

Step 11: Move each calculated pixel position (x,y) on to the elliptical path centered on (x_c , y_c) and plot the co-ordinates values

$x = x + x_c$,

$y = y + y_c$.

Step 12: Display the output points.

Step 13: Stop the program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
main()
{
                int gd=DETECT,gm;
                int xcenter,ycenter,rx,ry;
                int p,x,y,px,py,rx1,ry1,rx2,ry2;
                initgraph(&gd,&gm,"c:\\tc\\bgi");
                printf("Enter The Radius Value:\n");
                scanf("%d%d",&rx,&ry);
                printf("Enter The xcenter and ycenter Values:\n");
                scanf("%d%d",&xcenter,&ycenter);
                ry1=ry*ry;
                rx1=rx*rx;
                ry2=2*ry1;
                rx2=2*rx1;
/* REGION 1 */
                x=0;
                y=ry;
                plotpoints(xcenter,ycenter,x,y);
                p=(ry1-rx1*ry+(0.25*rx1));
                px=0;
```

```
                py=rx2*y;
                while(px<py)
                {
                        x=x+1;
                        px=px+ry2;
                        if(p>=0)
                                y=y-1;
                                py=py-rx2;
                        if(p<0)
                                 p=p+ry1+px;
                         else
                                 p=p+ry1+px-py;
                plotpoints(xcenter,ycenter,x,y);
/* REGION 2*/
                p=(ry1*(x+0.5)*(x+0.5)+rx1*(y-1)*(y-1)-rx1*ry1);
                while(y>0)
                {
                        y=y-1;
                        py=py-rx2;
                        if(p<=0)
                        {
                                x=x+1;
                                px=px+ry2;
                       }
                       if(p>0)
```

```
                                        p=p+rx1-py;
                                else
                                        p=p+rx1-py+px;
                            plotpoints(xcenter,ycenter,x,y);
                           }
                        }
                            getch();
                            return(0);
        }

int plotpoints(int xcenter,int ycenter,int x,int y)
{
                            putpixel(xcenter+x,ycenter+y,6);
                            putpixel(xcenter-x,ycenter+y,6);
                            putpixel(xcenter+x,ycenter-y,6);
                            putpixel(xcenter-x,ycenter-y,6);

}
```

OUTPUT:

Enter the Radius Value : 10 30

Enter the X Center and Y Center: 300 150

RESULT:

Thus the above program has been executed and output is verified.

OUTPUT PRIMITIVES

EX NO: 5

Aim :

To write a C program to draw the various attributes of line, circle and ellipse.

Algorithm:

Step 1: Start the program.

Step 2: Initialize the variables.

Step 3: Call the initgraph() function

Step 4: Set color for the output primitives.

Step 5: Using Outtextxy() display the chosen particular primitives.

Step 6: Include the various attributes of line, circle and ellipse.

Step 7: close the graph and run the program.

Step 8: stop the program

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<string.h>
void main()
{
        char ch='y';
        int gd=DETECT,gm,x1,y1,x2,y2,rad,sa,ea,xrad,yrad,i;
        initgraph(&gd,&gm,"");
        while(ch=='y')
        {
                cleardevice();
                setbkcolor(9);
                outtextxy(100,130,"Choose From The Following ");
                outtextxy(100,150,"1. Line");
                outtextxy(100,170,"2.Circle");
                outtextxy(100,190,"3.Box");
                outtextxy(100,210,"4.Arc");
                outtextxy(100,230,"5.Ellipse");
                outtextxy(100,250,"6.Rectangle");
                outtextxy(100,270,"7.Exit");
                ch=getch();
```

```
cleardevice();
switch(ch)
{
   case '1':
         line(100,200,300,400);
         break;
   case '2':
         circle(200,200,100);
         break;
   case '3':
         setfillstyle(5,4);
         bar(100,300,200,100);
         break;
   case '4':
         setfillstyle(5,4);
         arc(200,200,100,300,100);
         break;
   case '5':
         setfillstyle(5,4);
         fillellipse(100,100,50,100);
         break;
case '6':
     settextstyle(DEFAULT_FONT,0,2);
      outtextxy(120,140,"AMS COLLEGE ");
      line(100,100,100,300);
```

```
                    line(300,300,100,300);
                    line(100,100,300,100);
                    line(300,100,300,300);
                    break;
                case '7':
                    closegraph();
                    return;
        }
        ch='y';
        getch();
        }
}
```

OUTPUT:

Choose from the following

1.Line

2. Circle

3.Box

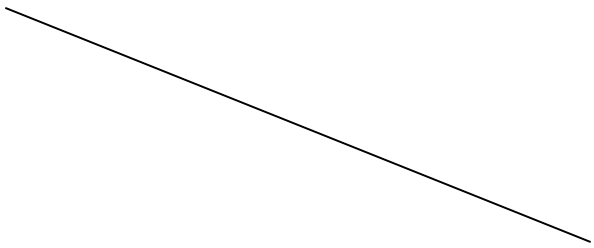
4.Arc

5.Ellipse

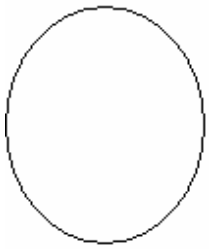
6.Rectangle

7.Exit

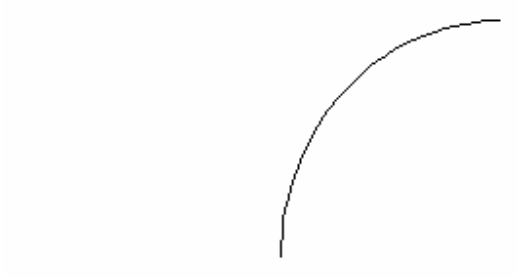
LINE



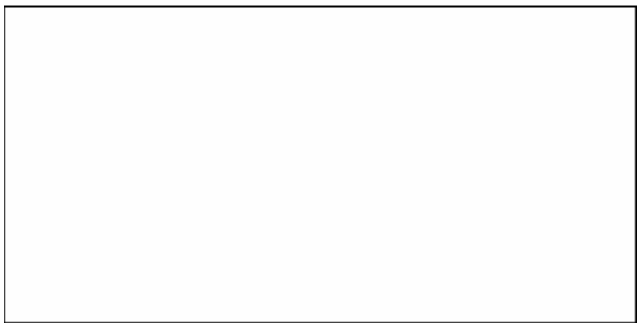
CIRCLE



ARC



RECTANGLE



RESULT:

Thus the above program has been executed and output is verified.

TWO – DIMENSIONAL TRANSFORMATION

EX NO: 6

Aim :

To write a C program to perform 2D transformations such as translation, rotation, scaling, reflection and shearing.

Algorithm:

Step 1: Start the program.

Step 2: Input the object coordinates

Step 3: For Translation

a) Enter the translation factors tx and ty.
b) Move the original coordinate position (x,y) to a new position (x1,y1).ie. x=x+x1, y=y+y1.
c) Display the object after translation

Step 4: For Rotation

a) Enter the radian for rotation angle θ.
b) Rotate a point at position (x,y) through an angle θ about the origin x1=xcos θ - ysin θ , y1=ycos θ + xsin θ.
c) Display the object after rotation

Step 5: For Scaling

a) Input the scaled factors sx and sy.
b) The transformed coordinates (x1,y1) , x1=x.sx and y1=y.sy.
c) Display the object after scaling

Step 6: For Reflection

Reflection can be performed about x axis and y axis.
a) Reflection about x axis : The transformed coordinates are x1=a and y1=-y.
b) Reflection about y axis : The transformed coordinates are x1=x and y1=y.
c) Display the object after reflection

Step 7: For Shearing

a) Input the shearing factors shx and shy.

b) Shearing related to x axis : Transform coordinates x1=x+shx*y and y1=y.
c) Shearing related to y axis : Transform coordinates
x1=x and y1=y+shy*x.
d) Input the xref and yref values.
e) X axis shear related to the reference line y-yref is
x1=x+shx(y-yref) and y1=y.
f) Y axis shear related to the reference line x=xref is
x1=x
g) Display the object after shearing

Step 8: Stop the Program.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<dos.h>
#include<math.h>
#include<stdlib.h>
void menu();
void input();
void output();
void translation();
void rotation();
void scaling();
void shearing();
void reflection();
        int a[10][2],i,x,option,temp,angle,tx,ty,fx,fy,sh,k,n,axis,y;
        float sx,sy;

void menu()
{
                printf("menu\n");
                printf("1.Translation\n");
                printf("2.rotation\n");
                printf("3.scaling\n");
                printf("4.shearing\n");
```

```
printf("5.reflection\n");
printf("6.exit\n");
printf("enter the choice:");
scanf("%d",&option);
switch(option)
{
        case  1:
                input();
                translation();
                break;
        case 2:
                input();
                rotation();
                break;
        case 3:
                input();
                scaling();
                break;

        case 4 :
                input();
                shearing();
                break;
        case 5:
                input();
```

```
                        reflection();
                        break;
                case 6:
                        exit(0);
                        break;
        }
}
void input()
{
        printf("enter the number of vertices:" );
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("enter the coordinates:");
                scanf("%d%d%d%d",&a[i][0],&a[i][1],&a[i+1][0],&a[i+1][1]);
        }
}

void output()
{
        cleardevice();
        for(i=0;i<n;i++)
        {
                line(a[i][0],a[i][1],a[i+1][0],a[i+1][1]);
        }
```

```
}
void translation()
{
        output();
        printf("enter the tranformation vertex tx,ty:\n");
        scanf("%d%d",&tx,&ty);
        for(i=0;i<=n;i++)
        {
            a[i][0]=a[i][0]+tx;
            a[i][1]=a[i][1]+ty;
        }
        output();
        delay(10);
        menu();
}

void rotation()
{
        output();
        printf("enter the rotating angle:");
        scanf("%d",&y);
        printf("enter the pivot point:");
        scanf("%d%d",&fx,&fy);
        k=(y*3.14)/180;
        for(i=0;i<=n;i++)
```

```
                {
                        a[i][0]=fx+(a[i][0]-fx)*cos(k)-(a[i][1]-fy)*sin(k);
                        a[i][1]=fy+(a[i][0]-fx)*sin(k)-(a[i][1]-fy)*cos(k);
                }
                output();
                delay(10);
                menu();
}
void scaling()
{
                output();
                printf("enter the scaling factor\n");
                scanf("%f%f",&sx,&sy);
                printf("enter the fixed point:");
                scanf("%d%d",&fx,&fy);
                for(i=0;i<=n;i++)
                {
                        a[i][0]=a[i][0]*sx+fy*(1-sx);
                        a[i][1]=a[i][1]*sy+fy*(1-sy);
                }
                output();
                delay(10);
                menu();
}
void shearing()
```

```
{
        output();
        printf("enter the shear value:");
        scanf("%d",&sh);
        printf("enter the fixed point:");
        scanf("%d%d",&fx,&fy);
        printf("enter the axis for shearing if x-axis then 1 if y-axis the      0:");
        scanf("%d",&axis);
        for(i=0;i<=n;i++)
        {
            if(axis==1)
            {
                a[i][0]=a[i][0]+sh*(a[i][1]-fy);
            }
            else
            {
                a[i][1]=a[i][1]+sh*(a[i][0]-fx);
            }
        }
        output();
        delay(10);
        menu();
}
void reflection()
{
```

```
                output();
                for(i=0;i<=n;i++)
                {
                        temp=a[i][0];
                        a[i][0]=a[i][1];
                        a[i][1]=temp;
                }
                output();
                delay(10);
                menu();
}
void main()
{
                int gd=DETECT,gm;
                initgraph(&gd,&gm,"c:\\tc\\bgi");
                menu();
                getch();
}
```

OUTPUT:

Menu

Translation

1. Rotation
2. Rotation
3. Scaling
4. Shearing
5. Reflection
6. Exit

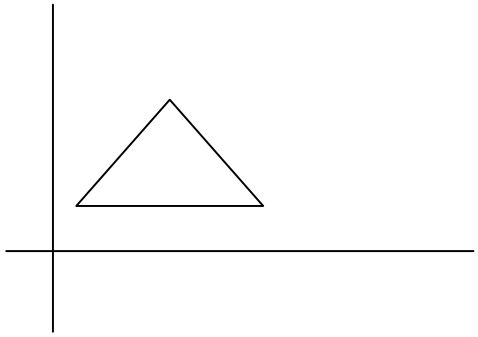
Enter the choice : 1

Enter the number of Vertices: 3

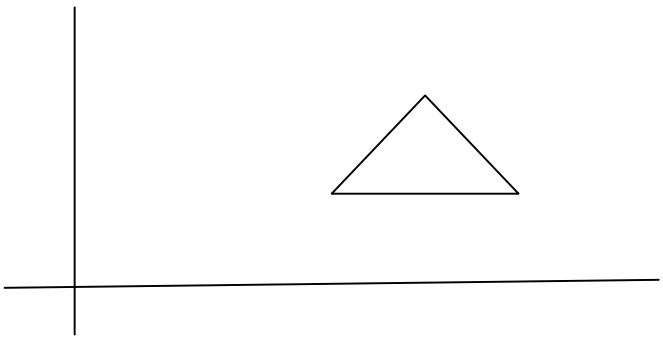
Enter the coordinates : 30 150 10 200

Enter the coordinates : 10 200 60 200

Enter the coordinates : 60 200 30 150



Enter the translation vector Tx, Ty : 90 60



ROTATION

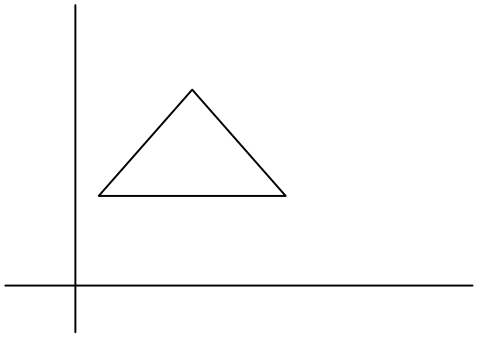
Enter the choice : 2

Enter the number of Vertices: 3

Enter the coordinates : 30 150 10 200

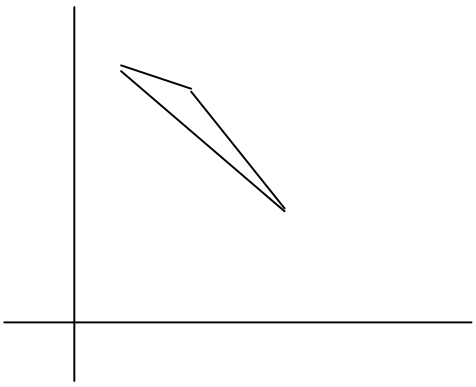
Enter the coordinates : 10 200 60 200

Enter the coordinates : 60 200 30 150



Enter the Rotating Angle : 90

Enter the Pivot Point : 100 200



SCALING

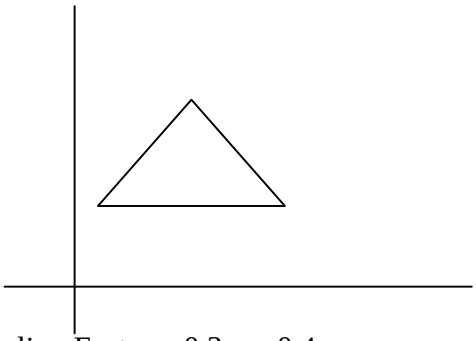
Enter the choice : 3

Enter the number of Vertices: 3

Enter the coordinates : 30 150 10 200

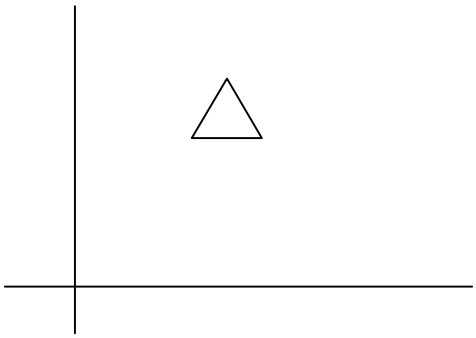
Enter the coordinates : 10 200 60 200

Enter the coordinates : 60 200 30 150



Enter the scaling Factor : 0.3 0.4

Enter the Fixed Point : 100 200



SHEARING

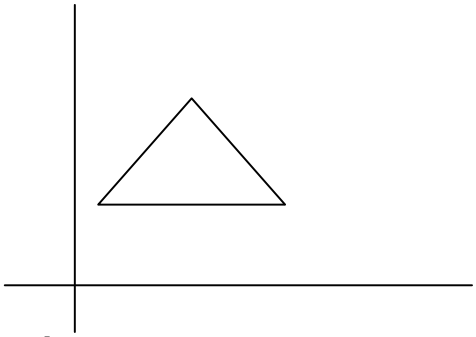
Enter the choice : 4

Enter the number of Vertices: 3

Enter the coordinates : 30 150 10 200

Enter the coordinates : 10 200 60 200

Enter the coordinates : 60 200 30 150

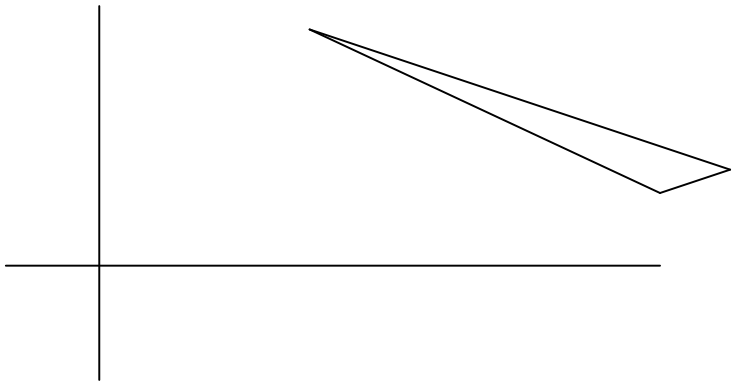


Enter the shear Value : 5

Enter the fixed point : 50 100

Enter the Axis for shearing if x-axis then 1

if y-axis then 0



REFLECTION

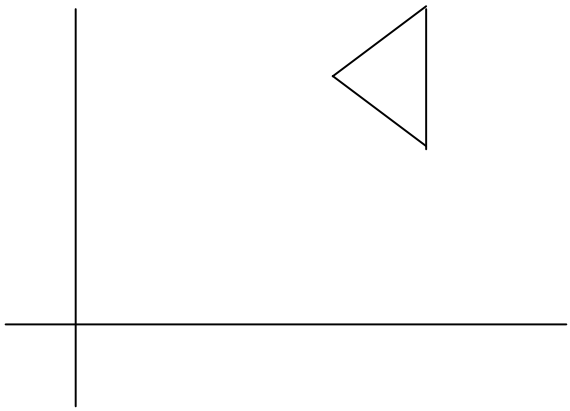
Enter the choice : 5

Enter the number of Vertices: 3

Enter the coordinates : 30 150 10 200

Enter the coordinates : 10 200 60 200

Enter the coordinates : 60 200 30 150



RESULT:

Thus the above program has been executed and output is verified.

COMPOSITE TWO – DIMENSIONAL TRANSFORMATION

EX NO: 7

Aim :

To write a C++ program to perform composite 2D transformations such as translation, rotation, scaling, reflection and shearing.

Algorithm:

Step 1: Start the program.

Step 2: Input the object coordinates**.**

Step 3: Translation

a) Enter the translation factors tx and ty.

b) Move the original coordinate position (x,y) to a new position

(x1,y1).ie. x=x+x1, y=y+y1.

Step 4: Rotationd) Enter the radian for rotation angle θ.

a) Rotate a point at position (x,y) through an angle θ about the origin x1=xcos θ - ysinθ , y1=ycosθ + xsinθ.

Step 5: Scaling

a) Input the scaled factors sx and sy.

b) The transformed coordinates (x1,y1) , x1=x.sx and y1=y.sy.

Step 6: Reflection

Reflection can be performed about x axis and y axis.

a) Reflection about x axis : The transformed coordinates are x1=a and y1=-y.

Step 7: Reflection about y axis : The transformed coordinates are x1=x and y1=y.

Step 8: Shearing

a) Input the shearing factors shx and shy.

b) Shearing related to x axis : Transform coordinates x1=x+shx*y and y1=y.

c) Shearing related to y axis : Transform coordinates x1=x and y1=y+shy*x.

d) Input the xref and yref values.

e) X axis shear related to the reference line y-yref is

x1=x+shx(y-yref) and y1=y.

f) Y axis shear related to the reference line x=xref is x1=x and y1=y+shy(x-xref)

Step 9: Finally display the transformed object after the successive transformations.

Step 10: Stop the Program.

PROGRAM:

```
#include<iostream.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
#include<stdlib.h>
void main()
{
        int  gd,gm,n,i,xa[10],ya[10],op,tx,ty,xa1[10],ya1[10],theta,xf,yf,rx,ry,
       sx,sy,shx,shy,xref,yref;
       char d;
       gd=DETECT;
       initgraph(&gd,&gm,"");
       cout<<"enter the no of points";
       cin>>n;
       for(i=0;i<n;i++)
       {
              cout<<"enter the coordinates"<<i+1;
              cin>>xa[i]>>ya[i];
       }

       do
       {
              cout<<"menu";
              cout<<"\n1.translation\n2.rotation\n3.scaling\n4.shearing\n5.reflection\n6.exit";
              cin>>op;
              switch(op)
              {
               case 1:
                     cout<<"enter the translation vector";
                     cin>>tx>>ty;
                     for(i=0;i<n;i++)
                     {
                            xa1[i]=xa[i]+tx;
                            ya1[i]=ya[i]+ty;
                     }
                     cleardevice();
                     cout<<"before translation";
                     for(i=0;i<n;i++)
                     {
                            line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);
                     }
                     cout<<"after translation";
                     for(i=0;i<n;i++)
                     {
                            line(xa1[i],ya1[i],xa1[(i+1)%n],ya1[(i+1)%n]);
```

```
}
getch();
cleardevice();
break;
case 2:
        cout<<"enter the rotation angle";
        cin>>theta;
        theta=(theta*3.14)/180;
        cout<<"enter the reference points";
        cin>>xf>>yf;
        for(i=0;i<n;i++)
        {
                xa1[i]=xf+(xa[i]-xf)*cos(theta)-(ya[i]-yf)*sin(theta);
                ya1[i]=yf+(xa[i]-xf)*sin(theta)-(ya[i]-yf)*cos(theta);
        }
        cleardevice();
        cout<<"before rotation";
        for(i=0;i<n;i++)
        {
                line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);
        }
        cout<<"after rotation";
        for(i=0;i<n;i++)
        {
                line(xa1[i],ya1[i],xa1[(i+1)%n],ya1[(i+1)%n]);
        }
        getch();
        cleardevice();
        break;
case 3:
        cout<<"enter the scaling factor";
        cin>>sx>>sy;
        cout<<"enter the reference point";
        cin>>rx>>ry;
        for(i=0;i<n;i++)
        {
                xa1[i]=xa[i]*sx+rx*(1-sx);
                ya1[i]=ya[i]*sy+ry*(1-sy);
         }
        cleardevice();
        cout<<"before scaling";
        for(i=0;i<n;i++)
        {
                line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);
        }
         cout<<"after scaling";
```

```
 for(i=0;i<n;i++)
{
        line(xa1[i],ya1[i],xa1[(i+1)%n],ya1[(i+1)%n]);
}
getch();
cleardevice();
break;
case 4:
cout<<"enter the shear value";
cin>>shx>>shy;
cout<<"enter the reference point";
cin>>xref>>yref;
cout<<"enter the shear direction x or y";
cin>>d;
if(d=='x')
{
        for(i=0;i<n;i++)
        {
                xa1[i]=xa[i]+shx*(ya[i]-yref);
                ya1[i]=ya[i];
         }
}
cleardevice();
cout<<"before shearing";
for(i=0;i<n;i++)
{
        line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);
}
cout<<"after shearing";
for(i=0;i<n;i++)
{
        line(xa1[i],ya1[i],xa1[(i+1)%n],ya1[(i+1)%n]);
}
getch();
cleardevice();
break;
case 5:
cout<<"before reflection";
for(i=0;i<n;i++)
{
        line(xa[i],ya[i],xa[(i+1)%n],ya[(i+1)%n]);
}
 cout<<"after reflection";
for(i=0;i<n;i++)
{
        line(ya[i],xa[i],ya[(i+1)%n],xa[(i+1)%n]);
```

```
                                        }
                                        getch();
                                        cleardevice();
                                        break;
                                case 6:
                                        exit(0);
                                        break;
                        }
        }while(op!=6);
}
```

OUTPUT :

enter the no of points: 3
enter the coordinates 1: 50 150
enter the coordinates 2: 50 50
enter the coordinates 3: 75 150

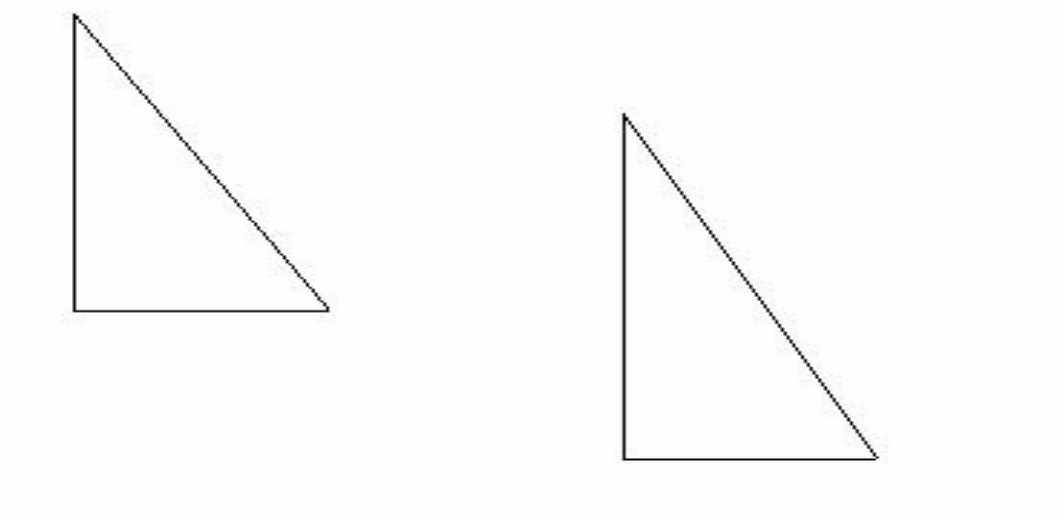
menu

1. translation
2. rotation
3. scaling
4.shearing
5.reflection
6.exit

1

enter the translation vector: 30 40

BEFORE TRANSLATION AFTER TRANSLATION



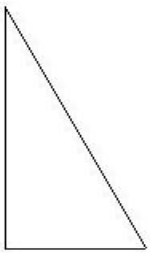
menu
1. translation
2. rotation
3. scaling
4.shearing
5.reflection
6.exit

2

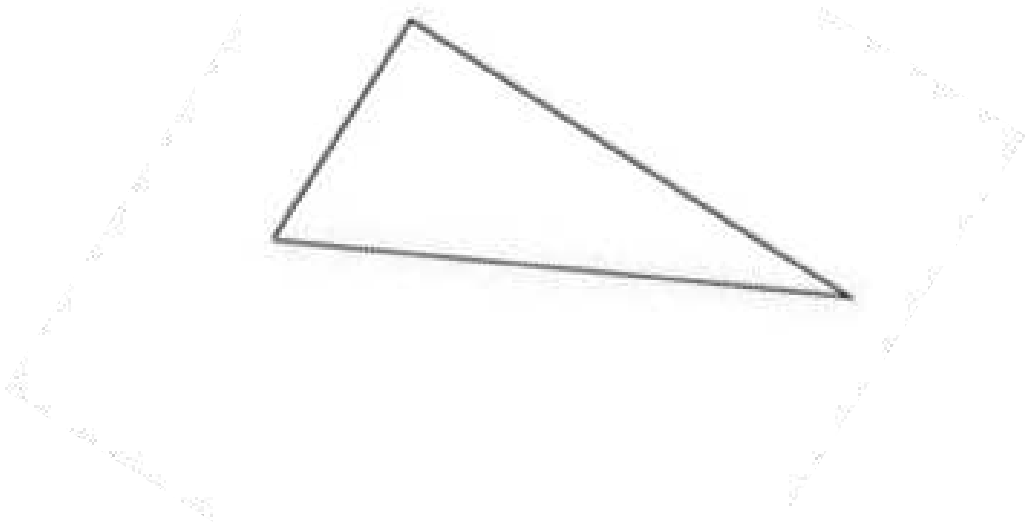
enter the rotation angle: 60

enter the reference points: 30 40

BEFORE



AFTER



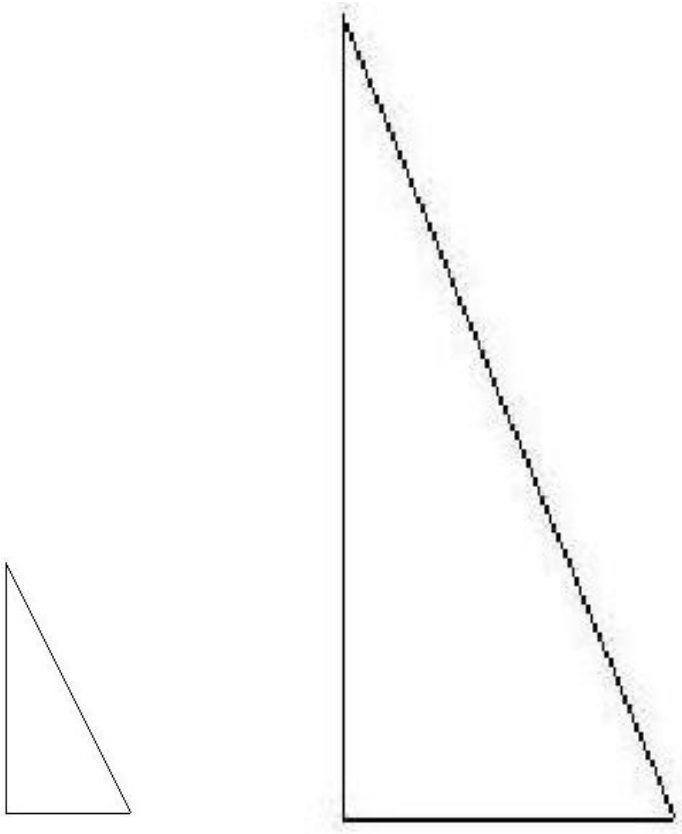
menu

1. translation
2. rotation
3. scaling
4.shearing
5.reflection
6.exit

3

Enter the scaling factor: 3 4
Enter the reference points: 30 40

BEFORE AFTER



menu
1. translation
2. rotation
3. scaling
4.shearing
5.reflection
6.exit

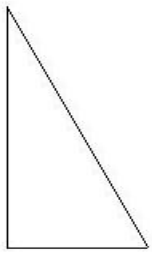
4

Enter the shear value: 3 4

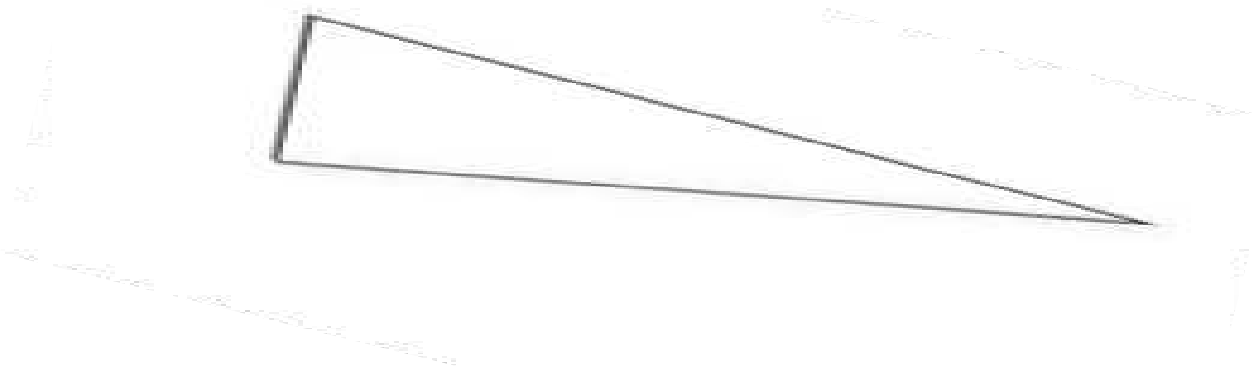
Enter the reference point: 20 30

Enter the shear direction x or y: X

BEFORE



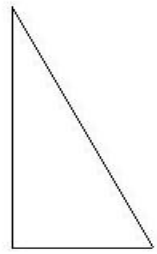
AFTER



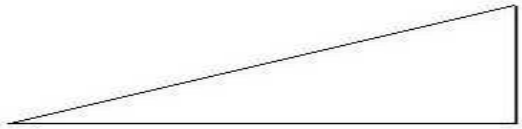
menu
1. translation
2. rotation
3. scaling
4.shearing
5.reflection
6.exit

5

BEFORE



AFTER



menu
1. translation
2. rotation
3. scaling
4.shearing
5.reflection
6.exit

6

RESULT:

Thus the above program has been executed and output is verified.

COHEN SUTHERLAND 2D LINE CLIPPING

EX NO: 8

Aim :

To write a C program to clip a line using Cohen-Sutherland clipping algorithm.

Algorithm:

Step 1: Start the program.

Step 2: Enter the line end points and the window coordinates.

Step 3: Every line end point is assigned a code that identified the location of the point relative to the boundaries of the clipping rectangle.

Step 4: Check whether the line lies inside the window then it is entirely drawn.

Step 5: Check whether the line lies outside the window then it is entirely clipped.

Step 6: Otherwise check whether the line intersects the window:
a) Calculate differences between end points and clip boundaries.
b) Determine the intersection point and how much of the line is to be discarded.

Step 7: Display the Output.

Step 8: Stop the program.

PROGRAM:

```
#include<stdio.h>

#include<math.h>

#include<graphics.h>

#include<conio.h>

float cxl, cxr,cyt,cyb;
```

```
code(float,float);
void clip(float,float,float,float);
void rect(float,float,float,float);
void main()
{
        float x1,y1,x2,y2;
        int g=0,d;
        initgraph(&g,&d,"c:\\tc\\bin");
        settextstyle(1,0,1);
        outtextxy(40,15,"BEFORE CLIPPING");
        printf("\n\n\n please enter the left,bottom,right,top,of clip window");
        scanf("%f%f%f%f",&cxl,&cyt,&cxr,&cyt);
        rect(cxl,cyb,cxr,cyt);
        getch();
        printf("\n please enter the line(x1,y1,x2,y2):");
        scanf("%f%f%f%f",&x1,&y1,&x2,&y2);
        line(x1,y1,x2,y2);
        getch();
        cleardevice();
        settextstyle(1,0,1);
        outtextxy(40,15,"after clipping");
        clip(x1,y1,x2,y2);
        getch();
        closegraph();
}
```

```
void clip(float x1,float y1,float x2,float y2)
{
        int c,c1,c2;
        float x,y;
        c1=code(x1,y1);
        c2=code(x2,y2);
        getch();
        while((c1!=0)||(c2!=0))
        {
                if((c1&c2)!=0)
                        goto out;
                        c=c1;
                if(c==0)
                        c=c2;
                if((c&1)==1)
                {
                        y=y1+(y2-y1)*(cxl-x1)/(x2-x1);
                        x=cxl;
                }
                else
                        if((c&2)==2)
                        {
                            y=y1+(y2-y1)*(cxl-x1)/(x2-x1);
                            x=cxr;
                        }
```

```
else
if((c&8)==8)
{
        x=x1+(x2-x1)*(cyb-y1)/(y2-y1);
        y=cyb;
}
else
if((c&4)==4)
{
        x=x1+(x2-x1)*(cyt-y1)/(y2-y1);
        y=cyt;
}
if(c==c1)
{
        x1=x;
        y1=y;
        c1=code(x,y);
}

else
{
        x2=x;
        y2=y;
        c2=code(x,y);
}
```

```
        }
        out : rect(cxl,cyb,cxr,cyt);
        line(x1,y1,x2,y2);
}
code(float x,float y)
{
        int c=0;
        if(x<cxl)
                c=1;
        else
                if(x>cxr)
                        c=2;
                if(y<cyb)
                        c=c|8;
                else
                if(y>cyt)
                        c=c|4;
                        return c;
}
void rect(float xl,float yb,float xr,float yt)
{
        line(xl,yb,xr,yb);
        line(xr,yb,xr,yt);
        line(xr,yt,xl,yt);
        line(xl,yt,xl,yb);
```

}

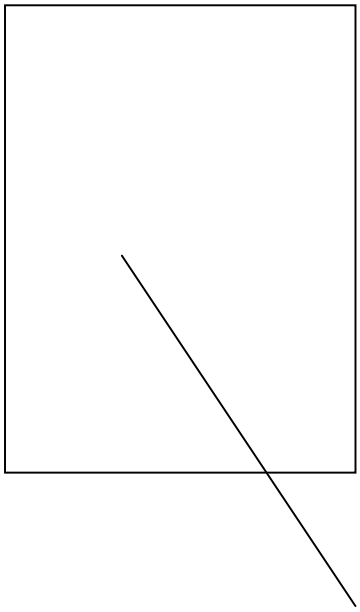
OUTPUT

Enter the left, bottom, right ,top of clip window
200
200
400
400

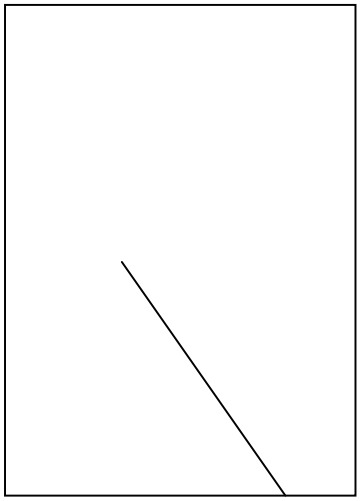
enter the line coordinates

250
300
400
450

Before Clipping:



After Clipping



RESULT:

Thus the above program has been executed and output is verified.

WINDOWING TO VIEWPORT MAPPING

EX NO: 9

Aim :

To write a C program to clip a Window to Viewport Mapping.

Algorithm:

Step 1: Start the program.

Step 2: get the maximum and minimum co-ordinates of the Window

Step 3: get the maximum and minimum co-ordinates of the ViewPort

Step 4: get the co-ordinates of a point by fitting window in viewport.

Step 5: Display the output.

Step 6: Stop theprorgam.

PROGRAM:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
main()
 {
                float sx,sy;
                int w1,w2,w3,w4,x1,x2,x3,x4,y1,y2,y3,y4,v1,v2,v3,v4;
                int gd=DETECT,gm;
                initgraph(&gd,&gm,"c:\\tc\\bgi");
                printf("Enter The Coordinate x1,y1,x2,y2,x3,y3\n");
                scanf("%d%d%d%d%d%d",&x1,&y1,&x2,&y2,&x3,&y3);
                cleardevice();
                w1=5;
                w2=5;
                w3=635;
                w4=465;
                rectangle(w1,w2,w3,w4);
                line(x1,y1,x2,y2);
                line(x2,y2,x3,y3);
                line(x3,y3,x1,y1);
                getch();
                v1=425;
                v2=75;
```

```
                v3=550;
                v4=250;
                sx=(float)(v3-v1)/(w3-w1);
                sy=(float)(v4-v2)/(w4-w2);
                rectangle(v1,v2,v3,v4);
                x1=v1+floor(((float)(x1-w1)*sx)+.5);
                x2=v1+floor(((float)(x2-w1)*sx)+.5);
                x3=v1+floor(((float)(x3-w1)*sx)+.5);
                y1=v2+floor(((float)(y1-w2)*sy)+.5);
                y2=v2+floor(((float)(y2-w2)*sy)+.5);
                y3=v2+floor(((float)(y3-w2)*sy)+.5);
                line(x1,y1,x2,y2);
                line(x2,y2,x3,y3);
                line(x3,y3,x1,y1);
                getch();
                return 0;
}
```

OUTPUT

Enter The Coordinate x1,y1,x2,y2,x3,y3

100

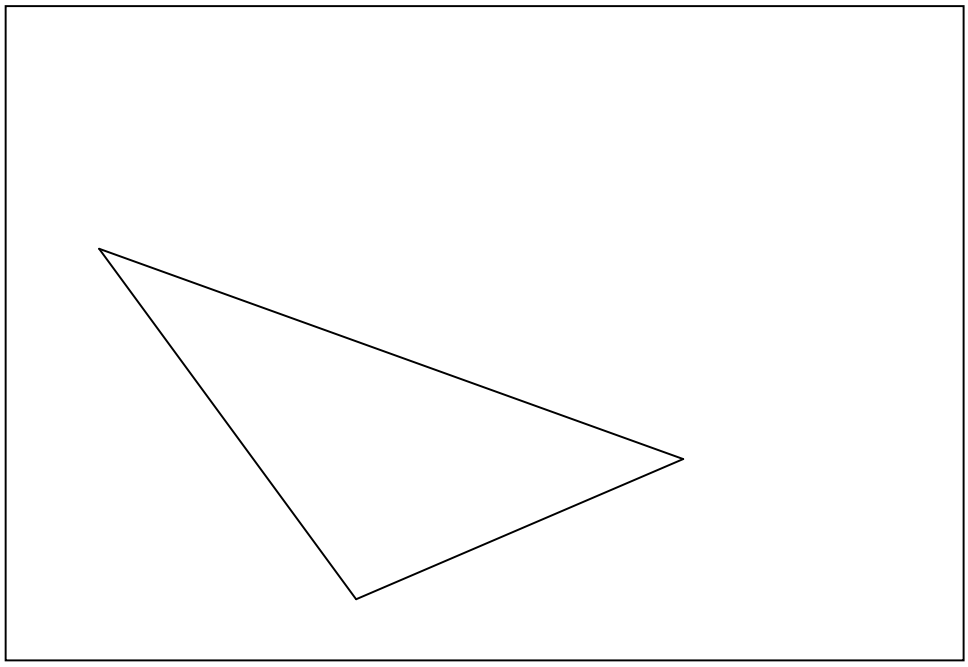
200

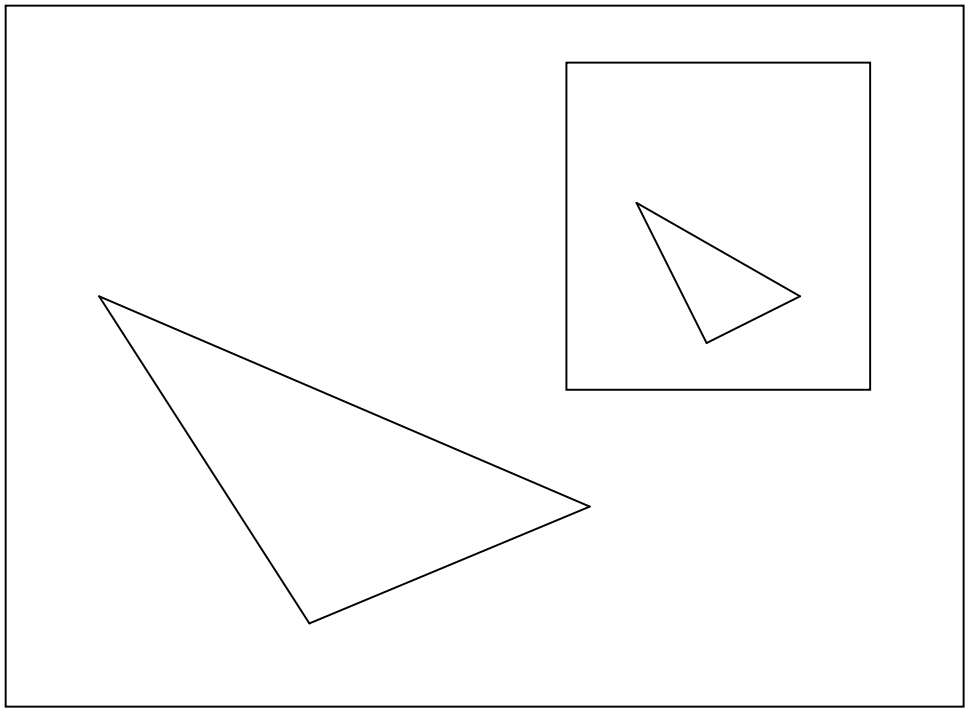
300

400

500

350





RESULT:

Thus the above program has been executed and output is verified.

SUTHERLAND – HODGEMANN POLYGON CLIPPING ALGORITHM
EX NO: 10

Aim :

To write a C program to implement **SUTHERLAND – HODGEMANN** polygon clipping algorithm.

Algorithm:

Step 1: Start the program.

Step 2: Input Coordinates of all vertices of the polygon

Step 3: Input coordinates of the clipping window

Step 4: Consider the left edge of the window

Step 5: Compare the vertices of each edge of the polygon , individually with the clipping plane

Step 6: Save the resulting intersections and vertices in the new list of vertices according to four possible relationships between the edge and the clipping boundary discussed earlier

Step 7: Repeat the steps 4 and 5 for remaining edges of the clipping window. Each time the resultant list of vertices is successively passed to process the next edge of the clipping window

Step 8: Stop the Program.

PROGRAM:

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <math.h>
#include <process.h>
#define TRUE 1
#define FALSE 0
typedef unsigned int outcode;
outcode CompOutCode(float x,float y);
enum
{
        TOP = 0x1,
        BOTTOM = 0x2,
        RIGHT = 0x4,
        LEFT = 0x8
};
float xwmin,xwmax,ywmin,ywmax;
void clip(float x0,float y0,float x1,float y1)
{
        outcode outcode0,outcode1,outcodeOut;
        int accept = FALSE,done = FALSE;
        outcode0 = CompOutCode(x0,y0);
        outcode1 = CompOutCode(x1,y1);
        do
```

```
{
if(!(outcode0|outcode1))
{
        accept = TRUE;
        done = TRUE;
}
else
if(outcode0 & outcode1)
        done = TRUE;
else
{
        float x,y;
        outcodeOut = outcode0?outcode0:outcode1;
        if(outcodeOut & TOP)
        {
                x = x0+(x1-x0)*(ywmax-y0)/(y1-y0);
                y = ywmax;
        }
        else
        if(outcodeOut & BOTTOM)
        {
                x = x0+(x1-x0)*(ywmin-y0)/(y1-y0);
                y = ywmin;
        }
        else
```

```
            if(outcodeOut & RIGHT)
            {
                  y = y0+(y1-y0)*(xwmax-x0)/(x1-x0);
                  x = xwmax;
            }
            else
            {
                  y = y0+(y1-y0)*(xwmin-x0)/(x1-x0);
                  x = xwmin;
            }
            if(outcodeOut==outcode0)
            {
                  x0 = x;
                  y0 = y;
                  outcode0 = CompOutCode(x0,y0);
            }
            else
            {
                  x1 = x;
                  y1 = y;
                  outcode1 = CompOutCode(x1,y1);
            }
      }
}
while(done==FALSE);
```

```
        if(accept)
        line(x0,y0,x1,y1);
        outtextxy(150,20,"POLYGON AFTER CLIPPING");
        rectangle(xwmin,ywmin,xwmax,ywmax);
}
outcode CompOutCode(float x,float y)
{
        outcode code = 0;
        if(y>ywmax)
        code|=TOP;
        else
        if(y<ywmin)
                code|=BOTTOM;
        if(x>xwmax)
                code|=RIGHT;
        else
        if(x<xwmin)
                code|=LEFT;
                return code;
}
void main( )
{
float x1,y1,x2,y2;
/* request auto detection */
int gdriver = DETECT, gmode, n,poly[14],i;
```

```
clrscr( );
printf("Enter the no of sides of polygon:");
scanf("%d",&n);
printf("\nEnter the coordinates of polygon\n");
for(i=0;i<2*n;i++)
{
        scanf("%d",&poly[i]);
}
poly[2*n]=poly[0];
poly[2*n+1]=poly[1];
printf("Enter the rectangular coordinates of clipping window\n");
scanf("%f%f%f%f",&xwmin,&ywmin,&xwmax,&ywmax);
/* initialize graphics and local variables */
initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

outtextxy(150,20,"POLYGON BEFORE CLIPPING");
drawpoly(n+1,poly);
rectangle(xwmin,ywmin,xwmax,ywmax);
getch( );
cleardevice( );
for(i=0;i<n;i++)
clip(poly[2*i],poly[(2*i)+1],poly[(2*i)+2],poly[(2*i)+3]);
getch( );
restorecrtmode( );
}
```

OUTPUT:

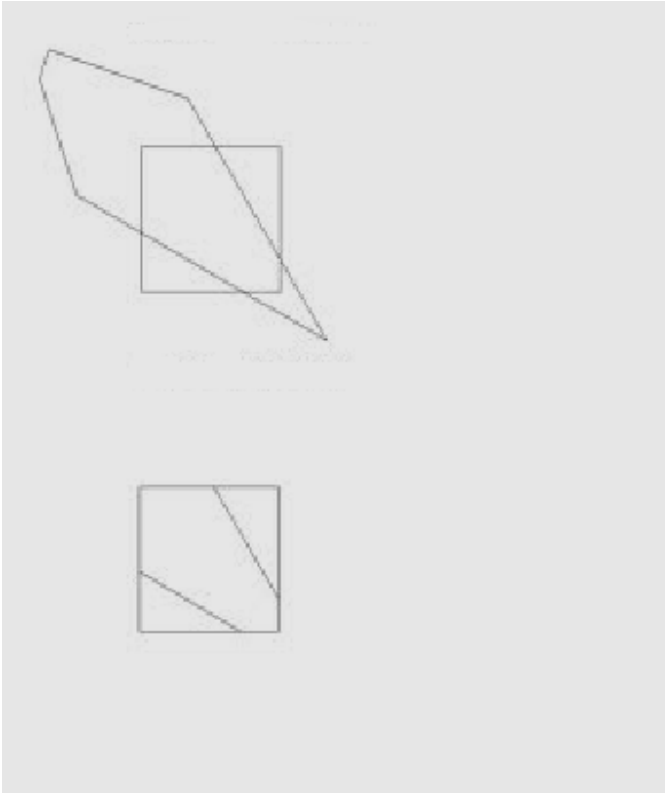
Enter the no of sides of polygon:5

Enter the coordinates of polygon
50
50
200
100
350
350
80
200
40
80

Enter the rectangular coordinates of clipping window
150
150
300
300

POLYGON CLIPPING

Before clipping



After clipping

RESULT:

Thus the above program has been executed and output is verified.

THREE – DIMENSIONAL TRANSFORMATION

EX NO: 11

Aim :

To write a C program to perform 3D transformations such as translation, rotation, scaling, reflection and shearing.

Algorithm:

Step 1: Start the program.

Step 2: Display the cube.

Step 3: input the translation vectortx,ty,tz.

Step 4: using the function line, display the object before and after translation.

Step 5: input the scaling factor and reference point.

Step 6: using the function line, display the object before and after scaling.

Step 7: input the rotation angle.

Step 8: using the function line,display the object before and after rotation.

Step 9: Stop the Program.

Program:

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
int maxx,maxy,midx,midy;
void axis()
{
        getch();
        cleardevice();
        line(midx,0,midx,maxy);
        line(0,midy,maxx,midy);
}

void main()
{
        int gd,gm,x,y,z,o,x1,x2,y1,y2;
        detectgraph(&gd,&gm);
        initgraph(&gd,&gm," ");
        setfillstyle(0,getmaxcolor());
        maxx=getmaxx();
        maxy=getmaxy();
        midx=maxx/2;
        midy=maxy/2;
        axis();
```

```
bar3d(midx+50,midy-100,midx+60,midy-90,5,1);
printf("Enter Translation Factor");
scanf("%d%d%d",&x,&y,&z);
axis();
printf("after translation");
bar3d(midx+(x+50),midy-(y+100),midx+x+60,midy-(y+90),5,1);
axis();
bar3d(midx+50,midy+100,midx+60,midy-90,5,1);
printf("Enter Scaling Factor");
scanf("%d%d%d",&x,&y,&z);
axis();
printf("After Scaling");
bar3d(midx+(x*50),midy-(y*100),midx+(x*60),midy-(y*90),5*z,1);
axis();
bar3d(midx+50,midy-100,midx+60,midy-90,5,1);
printf("Enter Rotating Angle");
scanf("%d",&o);
x1=50*cos(o*3.14/180)-100*sin(o*3.14/180);
y1=50*cos(o*3.14/180)+100*sin(o*3.14/180);
x2=60*sin(o*3.14/180)-90*cos(o*3.14/180);
y2=60*sin(o*3.14/180)+90*cos(o*3.14/180);
axis();
printf("After Rotation about Z Axis");
bar3d(midx+x1,midy-y1,midx+x2,midy-y2,5,1);
axis();
```

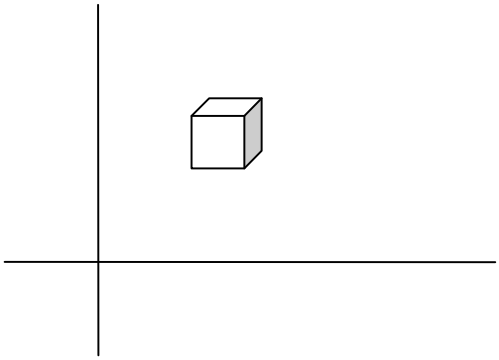
```
        printf("After Rotation about X Axis");
        bar3d(midx+50,midy-x1,midx+60,midy-x2,5,1);
        axis();
        printf("After Rotation about Y Axis");
        bar3d(midx+x1,midy-100,midx+x2,midy-90,5,1);
        getch();
        closegraph();
}
```

OUTPUT:

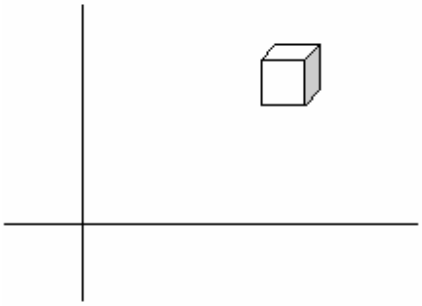
Translation

Enter Translation Factor : 50 60 70

Before Translation

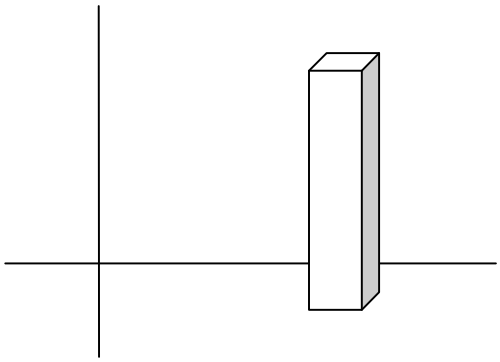


After Translation

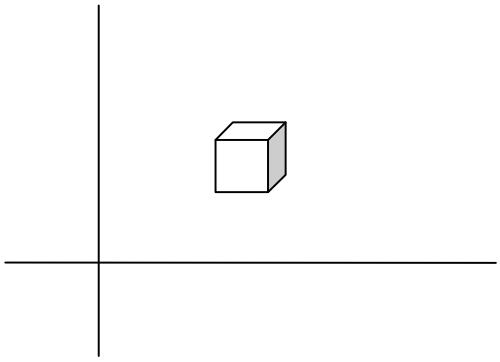


Scaling

Enter Scaling Factor : 80 90 95

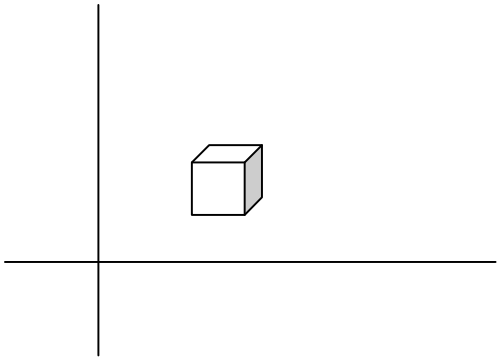


After Scaling

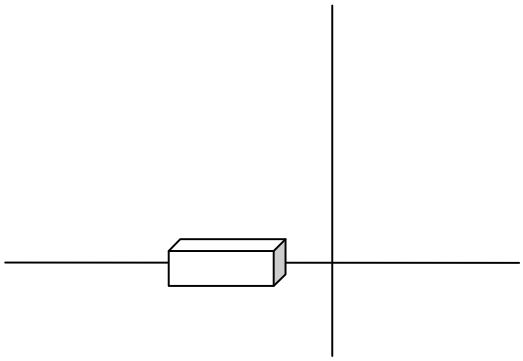


Rotation

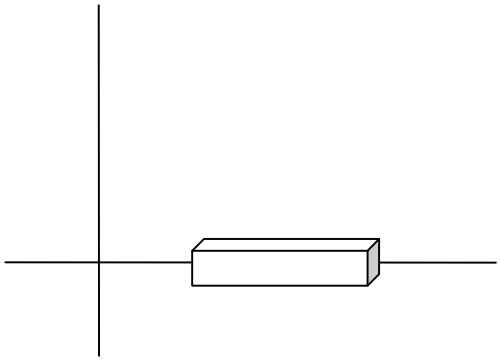
Enter Rotating Angle : 60



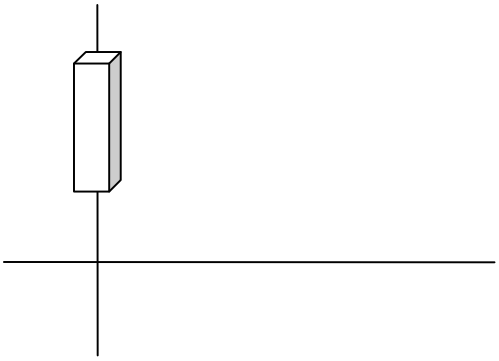
After Rotation about Z-Axis



After Rotation about X-Axis



After Rotation about Y-Axis :



RESULT:

Thus the above program has been executed and output is verified.

COMPOSITE THREE – DIMENSIONAL TRANSFORMATION

EX NO: 12

Aim :

To write a C++ program to perform composite 3D transformations such as translation, rotation, scaling, reflection and shearing.

Algorithm:

Step 1: Start the program.

Step 2: input the points for the cude and size of the Cube.

Step 3: Display the menu as 1.Translation 2.Scaling 3.Rotation 4.Exit

Step 4: Get the choice from the user.

Step 5: If the choice is 1 a point or an object is translated from position P to position P' with the operation P'=T.P where t_x,t_y and t_z specifying translation distances.

$x'=x+ t_x$,$y'=y+ t_y$,$z'=z+ t_z$

Step 6: If the choice is 2 the scaling transformation of a position P can be written as P'=S.P where scaling parameters s_x,s_y and s_z are assigned any positive values.

$x'=x.s_x$,$y'=y.s_y$,$z'=z.s_z$

Step 7: If the choice is 3 get the rotation angle. Rotate the figure with respect to the axis of rotation.

Step 8: About z axis rotation $x'=x\cos\Theta-y\sin\Theta$,$y'=x\sin\Theta+y\cos\Theta$

$z'=z$ Rotation can be expressed as $P'=R_z(\Theta).P$

Step 9: About x axis rotation $y'=y\cos\Theta-z\sin\Theta$ $z'=y\sin\Theta+z\cos\Theta$ $x'=x$

Rotation can be expressed as $P'=R_x(\Theta).P$

Step 10: About y axis rotation z'=zcosΘ-xsinΘ x'=zsinΘ+xcosΘ y'=y

Rotation can be expressed as $P'=R_y(\Theta).P$

Step 11: If choice is 4 exit the program.

Step 12: Stop the program.

PROGRAM:

```
#include<iostream.h>
#include<graphics.h>
#include<math.h>
#include<conio.h>
#include<stdlib.h>
class cube
{
 public:
 void drawcube(int x1[],int y1[])
 {
        int i;
        for(i=0;i<4;i++)
        {
        if(i<3)
        line(x1[i],y1[i],x1[i+1],y1[i+1]);
        line(x1[0],y1[0],x1[3],y1[3]);
 }
 for(i=4;i<8;i++)
 {
        if(i<7)
```

```
        line(x1[i],y1[i],x1[i+1],y1[i+1]);

        line(x1[4],y1[4],x1[7],y1[7]);

 }

 for(i=0;i<4;i++)

 {

        line(x1[i],y1[i],x1[i+4],y1[i+4]);

 }

}

};

void main()

{

        int
                i,x1[8],y1[8],x2[8],y2[8],z1[8],x3[8],y3[8],z3[8],x4[8],y4[8],theta,op,ch,tx,ty,tz,s
                x,sy,sz,xf,yf,zf,x,y,z,size;

        int driver=DETECT;

        int mode;

        initgraph(&driver,&mode,"C:\\tc++\bgi");

        cout<<"enter the points on the cube:";

        cin>>x>>y>>z;

        cout<<"enter the size of the edge:";

        cin>>size;

        x1[0]=x1[3]=x;
```

```
x1[1]=x1[2]=x+size;

x1[4]=x1[7]=x;

x1[5]=x1[6]=x+size;

y1[0]=y1[1]=y;

y1[2]=y1[3]=y+size;

y1[4]=y1[5]=y;

y1[6]=y1[7]=y+size;

z1[1]=z1[2]=z1[3]=z1[0]=z ;

z1[4]=z1[5]=z1[6]=z1[7]=z-size;

for(i=0;i<8;i++)

{

        x2[i]=x1[i]+z1[i]/2;

        y2[i]=y1[i]+z1[i]/2;

}

cube c;

getch();

cleardevice();

do

{

        cout<<"menu"<<endl;

        cout<<"\n1.translation\n2.rotation\n3.scaling\n4.exit\n";
```

```
cout<<"enter the choice:";
cin>>ch;
switch(ch)
{
case 1:
        cout<<"enter the translation vector:";
        cin>>tx>>ty>>tz;
        for(i=0;i<8;i++)
        {
                x3[i]=x1[i]+tx;
                y3[i]=y1[i]+ty;
                z3[i]=z1[i]+tz;
        }
        for(i=0;i<8;i++)
        {
                x4[i]=x3[i]+z3[i]/2;
                y4[i]=y3[i]+z3[i]/2;
        }
        cleardevice();
        cout<<"before translation";
        c.drawcube(x2,y2);
```

```
            getch();
            cleardevice();
            cout<<"after translation";
            c.drawcube(x4,y4);
            getch();
            cleardevice();
            break;
    case 2:
            cout<<"enter the rotation angle:";
            cin>>theta;
            theta=(theta*3.14)/180;
            cout<<"enter the direction"<<endl;
                cout<<"1.rotation about x axis"<<endl<<"2.rotation about y
            axis"<<endl<<"3.rotation about z axis";          cin>>op;
            if(op==1)
            {
                for(i=0;i<8;i++)
                {
                    x3[i]=x1[i];
                    y3[i]=y1[i]*cos(theta)-z1[i]*sin(theta);
                    z3[i]=y1[i]*sin(theta)+z1[i]*cos(theta);
```

```
            }
}
else
if(op==2)
{
 for(i=0;i<8;i++)
  {
    y3[i]=y1[i];
    x3[i]=z1[i]*cos(theta)-x1[i]*sin(theta);
    x3[i]=z1[i]*sin(theta)+x1[i]*cos(theta);
  }
}
else
if(op==3)
{
 for(i=0;i<8;i++)
 {
            z3[i]=z1[i];
            x3[i]=x1[i]*cos(theta)-y1[i]*sin(theta);
            y3[i]=x1[i]*sin(theta)+y1[i]*cos(theta);
 }
```

```
                }
                else
                cout<<"enter correct option";
                for(i=0;i<8;i++)
                {
                        x4[i]=x3[i]+z3[i]/2;
                        y4[i]=y3[i]+z3[i]/2;
                }
                cleardevice();
                cout<<"before rotation";
                c.drawcube(x2,y2);
                getch();
                cleardevice();
                cout<<"after rotation";
                c.drawcube(x4,y4);
                getch();
                cleardevice();
                break;
        case 3:
                cout<<"enter the scaling factor:";
                cin>>sx>>sy>>sz;
```

```
cout<<"enter the reference point:";
cin>>xf>>yf>>zf;
for(i=0;i<8;i++)
{
        x3[i]=xf+(x1[i]*sx)+xf*(1-sx);
        y3[i]=yf+(y1[i]*sy)+yf*(1-sy);
        z3[i]=zf+(z1[i]*sz)+zf*(1-sz);
 }
for(i=0;i<8;i++)
{
        x4[i]=x3[i]+z3[i]/2;
        y4[i]=y3[i]+z3[i]/2;
}
cleardevice();
cout<<"before scaling";
c.drawcube(x2,y2);
getch();
cleardevice();
 cout<<"after scaling";
 c.drawcube(x4,y4);
getch();
```

```
                cleardevice();
                break;
            case 4:
                exit(0);
                break;
        }
    }
    while(op!=4);
    getch();
}
```

OUTPUT :

Enter the points in the cube : 100 100 100

Enter the Size of Edge : 50

MENU

1.Translation

2.Rotation

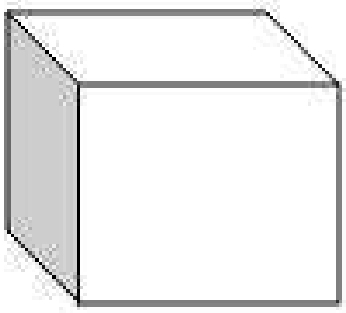
3.scaling

4.exit

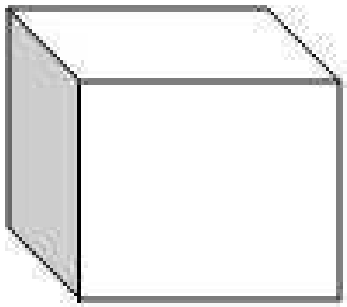
Enter your choice : 1

Enter the Translation Vector 5 10 15

Before



After



Enter your Choice : 2

Rotation

Enter the Rotation Angle : 60

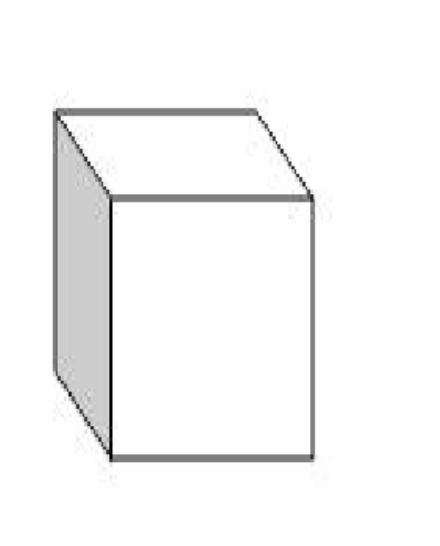
Enter the Direction

1.Rotation about x-axis

2.Rotation about y-axis

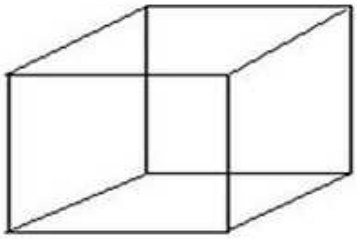
3.Rotation about z-axis

Before :

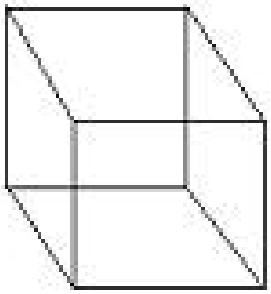


After :

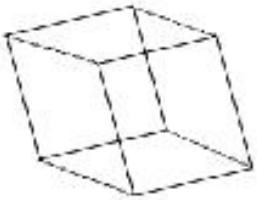
Rotation about x-axis



Rotation about y-axis



Rotation about z-axis



MENU

1.Translation

2.Rotation

3.scaling

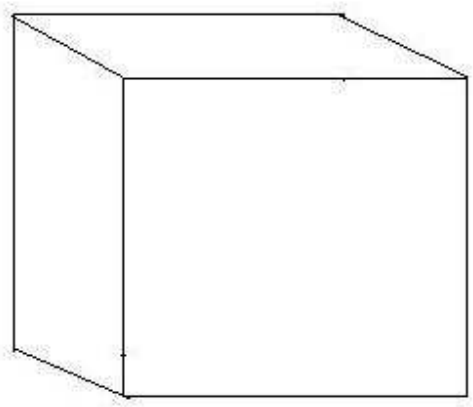
4.exit

Enter your choice : 3

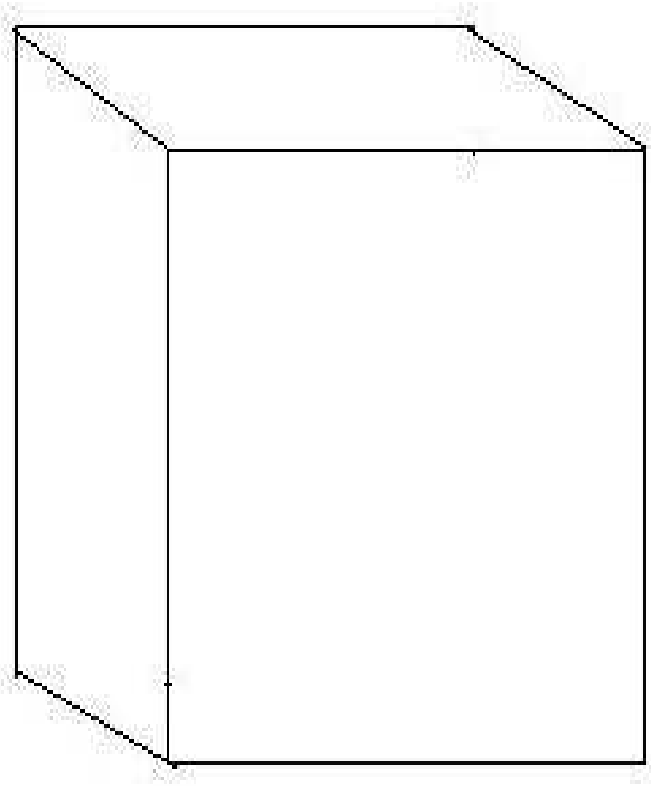
Enter the Scaling Factor : 30 40 50

Enter the Reference point : 20 35 45

BEFORE



AFTER



MENU

1.Translation

2.Rotation

3.scaling

4.exit

Enter your choice : 4

RESULT:

Thus the above program has been executed and output is verified.

VISUALIZING PROJECTIONS OF 3D IMAGES

EX NO: 13

Aim :

To write a C program for implementation of Visualizing Projections Of 3d Images.

Algorithm:

Step 1: Start the program.

Step 2: input the number of edges.

Step 3: input the start pt. and end pt. for the all the edges.

Step 4: draw and display the image obtained from the these points.

Step 5: generate the top view and display it.

Step 6: generate the side view and display it.

Step 7: generate the front view and display it.

Step 8: Stop the program.

PROGRAM :

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

struct point
{
        int x,y,z;
};
struct edge
{
         struct point start;
         struct point end;
};
float pi=3.14/180.0;
void convert2d(int *x,int *y,int *z)
{
        int xp,yp;
        xp=(*x)+(*z)*cos(pi*45)/tan(pi*45);
        yp=(*y)+(*z)*sin(pi*45)/tan(pi*45);
        *x=xp;
        *y=yp;
}
```

```
void screen(int *x,int *y)
{
        int xm,ym;
        xm=getmaxx();
        ym=getmaxy();
        *x=xm/2+*x;
        *y=ym/2-*y;
}
void draw3d(struct edge po[],int n)
{
        int i,x1,y1,z1,x2,y2,z2;
        for(i=0;i<n;i++)
        {
                x1=po[i].start.x;
                y1=po[i].start.y;
                z1=po[i].start.z;
                convert2d(&x1,&y1,&z1);
                x2=po[i].end.x;
                y2=po[i].end.y;
                z2=po[i].end.z;
                convert2d(&x2,&y2,&z2);
                screen(&x1,&y1);
                screen(&x2,&y2);
                line(x1,y1,x2,y2);
        }
```

```
}

void main()
{
        int gd=DETECT,gm=0;
        int i,tx,ty,tz,sx,sy,sz,n;
        int xx1,xx2,yy1,yy2;
        float rx,ry,rz;
        struct edge p[50],q[50],r[50],s[50],t[50],v[50];
        initgraph(&gd,&gm,"c:\\tc\\bgi");
        cleardevice();
        printf("\nEnter the number of edges:");
        scanf("%d",&n);
        for(i=0;i<n;i++)
        {
                printf("\nStart pt for edge %d(x,y,z):",i+1);
                scanf("%d%d%d",&p[i].start.x,&p[i].start.y,&p[i].start.z);
                printf("\nEnd pt for edge %d(x,y,z):",i+1);
                scanf("%d%d%d",&p[i].end.x,&p[i].end.y,&p[i].end.z);
        }
        cleardevice();
        printf("\n3D VIEW");
        draw3d(p,n);
        getch();
        cleardevice();
```

```
printf("\nTOP VIEW");
for(i=0;i<n;i++)
{
        xx1=p[i].start.x;
        yy1=p[i].start.z;
        xx2=p[i].end.x;
        yy2=p[i].end.z;
        screen(&xx1,&yy1);
        screen(&xx2,&yy2);
        line(xx1,yy1,xx2,yy2);
}
getch();
cleardevice();

printf("\nSIDE VIEW");
for(i=0;i<n;i++)
{
        xx1=p[i].start.z;
        yy1=p[i].start.y;
        xx2=p[i].end.z;
        yy2=p[i].end.y;
        screen(&xx1,&yy1);
        screen(&xx2,&yy2);
        line(xx1,yy1,xx2,yy2);
}
```

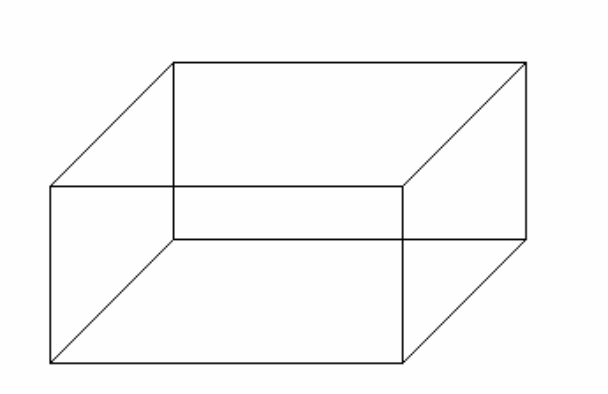
```
        getch();
        cleardevice();
        printf("\nFRONT VIEW");
        for(i=0;i<n;i++)
        {
                xx1=p[i].start.x;
                yy1=p[i].start.y;
                xx2=p[i].end.x;
                yy2=p[i].end.y;
                screen(&xx1,&yy1);
                screen(&xx2,&yy2);
                line(xx1,yy1,xx2,yy2);
        }
        getch();
        cleardevice();
}
```

OUTPUT:

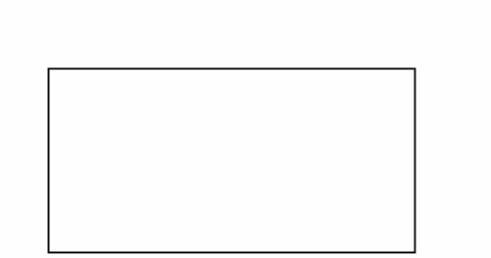
Start pt for edge 1 (x,y,z) :	0	0	0
End pt for edge 1 (x,y,z) :	200	0	0
Start pt for edge 2 (x,y,z) :	200	0	0
End pt for edge 2 (x,y,z) :	200	0	100
Start pt for edge 3 (x,y,z) :	200	0	100
End pt for edge 3 (x,y,z) :	0	0	100
Start pt for edge 4 (x,y,z) :	0	0	100
End pt for edge 4 (x,y,z) :	0	0	0
Start pt for edge 5 (x,y,z) :	0	100	0
End pt for edge 5 (x,y,z) :	200	100	0
Start pt for edge 6 (x,y,z) :	200	100	0
End pt for edge 6 (x,y,z) :	200	100	100
Start pt for edge 7 (x,y,z) :	200	100	100
End pt for edge 7 (x,y,z) :	0	100	100
Start pt for edge 8 (x,y,z) :	0	100	100
End pt for edge 8 (x,y,z) :	0	100	0
Start pt for edge 9 (x,y,z) :	0	100	0
End pt for edge 9 (x,y,z) :	0	0	0
Start pt for edge 10 (x,y,z):	200	100	0
End pt for edge 10 (x,y,z) :	200	0	0
Start pt for edge 11 (x,y,z):	200	100	100
End pt for edge 11 (x,y,z) :	200	0	100
Start pt for edge 12 (x,y,z):	0	100	100

End pt for edge 12 (x,y,z) : 0 0 100

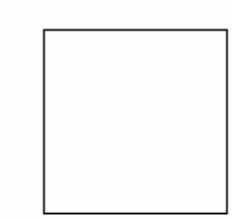
3D VIEW



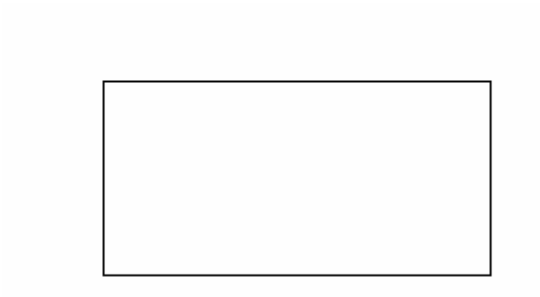
TOP VIEW



SIDE VIEW



FRONT VIEW



RESULT:

Thus the above program has been executed and output is verified.