Index

8.A	Write a program to implement Cohen-Sutherland clipping.	
8.B	Write a program to implement Liang - Barsky Line Clipping Algorithm.	
9.A	Write a program to fill a circle using Flood Fill Algorithm.	
9.B	Write a program to fill a circle using Boundary Fill Algorithm.	
10.A	Develop a simple text screen saver using graphics functions.	
10.B	Perform smiling face animation using graphic functions.	
10.C	Draw the moving car on the screen.	

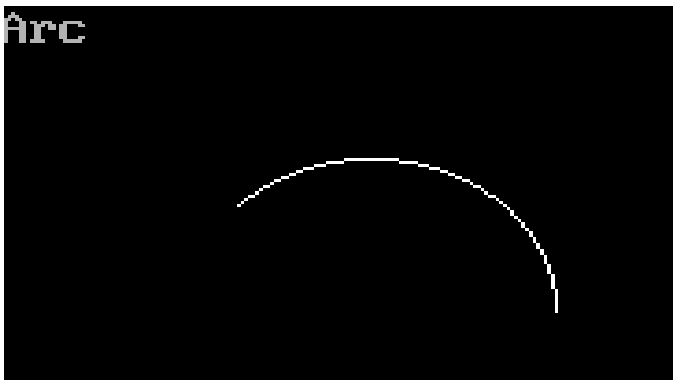
PRACTICAL 1

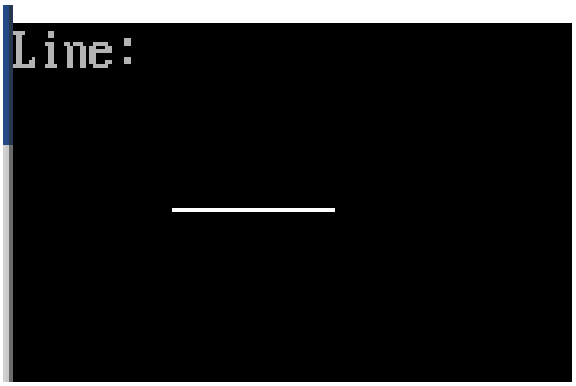
Aim: Write a program to enlist a basic function used for graphics in C/C++.Give example for each function.

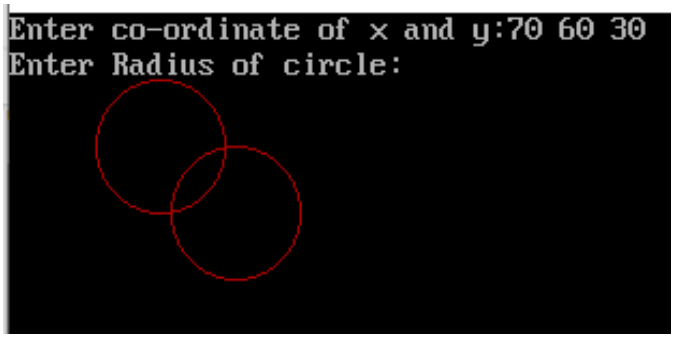
Code:

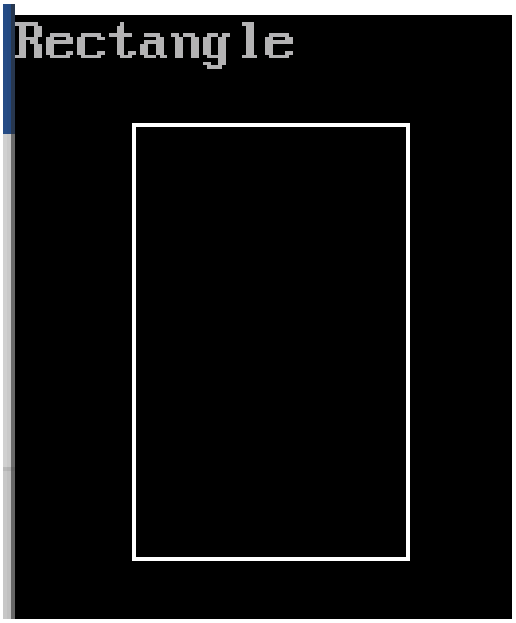
```
#include<iostream>
#include<conio.h>
#include<graphics.h>
using namespace std;
int main()
{
        int gd=DETECT,gm;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        cout<<"Line:";
        line(50,50,100,50);
        getch();
        cleardevice();
        // Rectangle
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        cout<<"Rectangle";
        rectangle(50,50,200,250);
        getch();
        cleardevice();
        //Arc
        initgraph(&gd,&gm,"C:TURBOC3\\BGI");
        cout<<"Arc";
        arc(100,100,0,135,50);
        getch();
        cleardevice();
        //circle
        int x,y,r;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        cout<<"Enter coordinate of x and y:";
        cin>>x>>y;
        cout<<"Enter Radius of circle:";
        cin>>r;
        setcolor(RED);
        circle(x,y,r);
        circle(x+x/2,y+y/2,r);
        getch();
        closegraph();
        return 0;
}
```

Output:









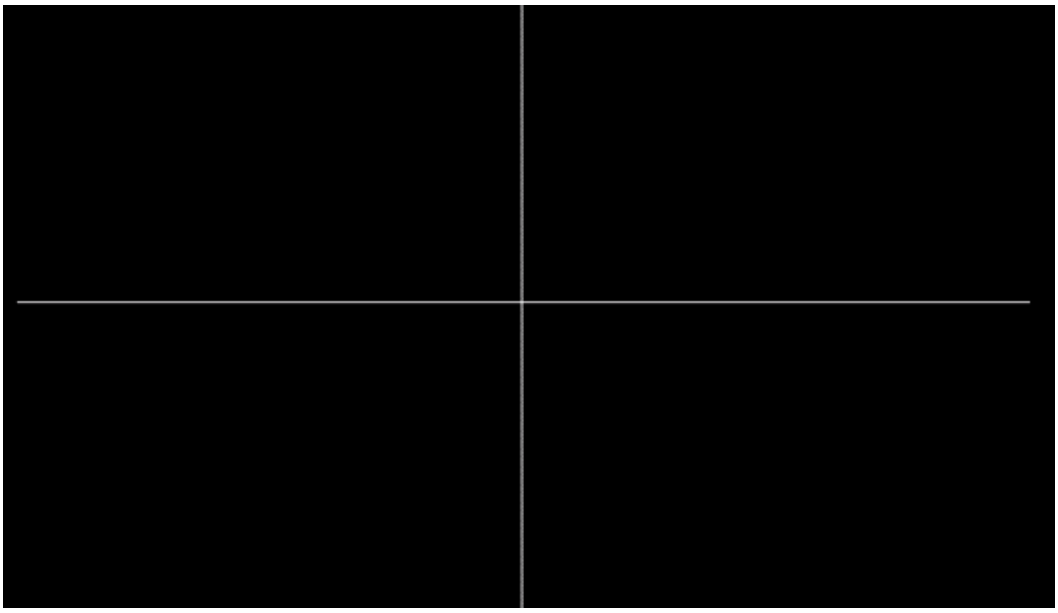
Aim: Draw a coordinate axis at the center of the screen.

Code:

```
#include<iostream.h>
```

```
#include<conio.h>
#include<graphics.h>
void main()
{
        int gd=DETECT,gm,midx,midy;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        cleardevice();
        midx=getmaxx()/2;
        midy=getmaxy()/2;
        line(1,midy,640,midy);
        line(midx,1,midx,480);
        getch();
        closegraph();
}
```

Output:



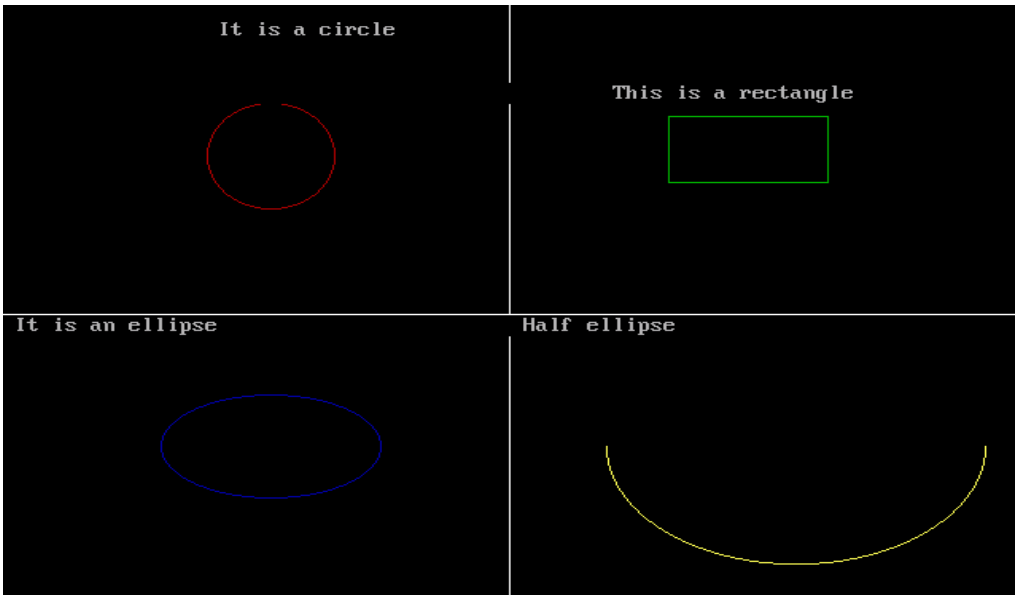
PRACTICAL 2

Aim: Divide your screen into four regions, Draw circle, rectangle,ellipse and half ellipse in each.

Code:

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
void main()
{
	int gd=DETECT,gm,midx,midy;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
	cleardevice();
	midx=getmaxx()/2;
	midy=getmaxy()/2;

	line(1,midy,640,midy);
	line(midx,1,midx,480);
	setcolor(RED);
	circle(midx+(-150),midy-(120),40);
	cout<<"\n\t\t It is a circle";
	setcolor(GREEN);
	rectangle(midx+(100),midy -(100),midx+(200),midy-(150));
	cout<<"\t\t\t\t  This is a rectangle \n\n\n\n";
	setcolor(BLUE);
	ellipse(midx+(-150),midy-(-100),0,360,midx+(-250),midy -(200));
	cout<<"\n\n\n\n\n\n\n\n It is an ellipse";
	setcolor(YELLOW);
	ellipse(midx+(180),midy-(-100),180,0,midx+(-200),midy-(150));
	cout<<"\n\n\n\n\n Half ellipse ";
	getch();
	closegraph();
}
```

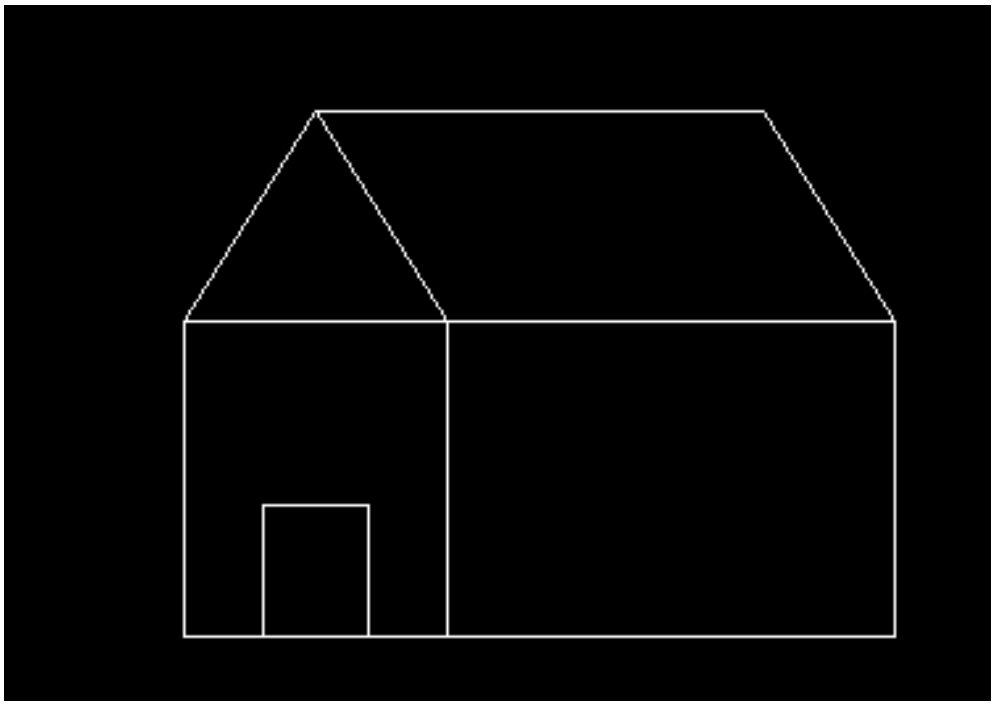
Output:



Aim: Draw a simple hut on the screen.

Code:

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
void main()
{
        int gd=DETECT,gm;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        setcolor(WHITE);
        rectangle(150,180,250,300);
        rectangle(250,180,420,300);
        rectangle(180,250,220,300);
        line(200,100,150,180);
        line(200,100,250,180);
        line(200,100,370,100);
        line(370,100,420,180);
        getch();
        closegraph();
}
```



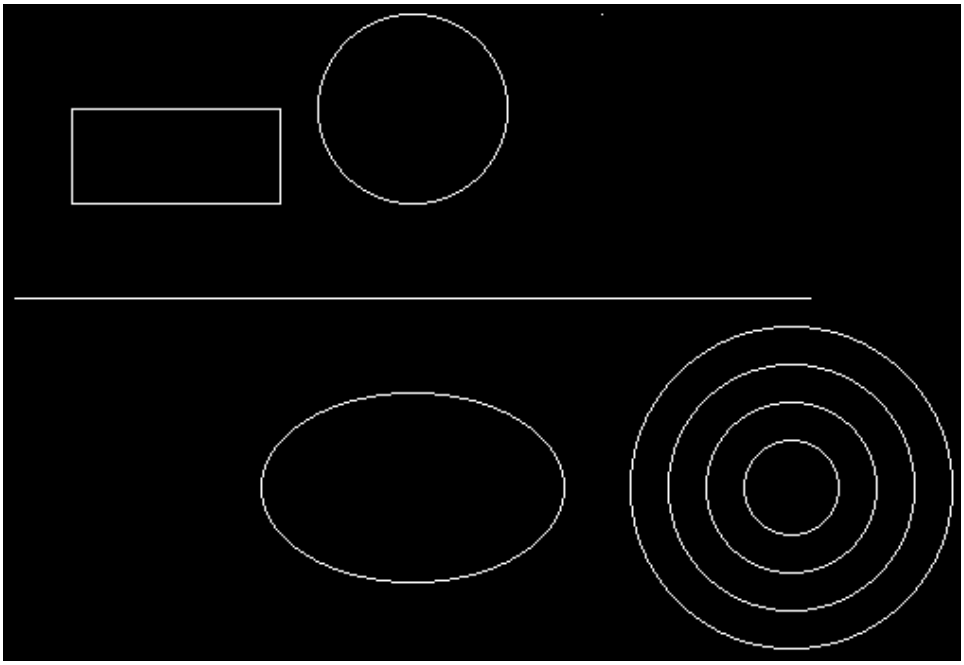
PRACTICAL 3

Aim: Draw the following basics shapes in the centers of the screen.
a.Circle b.Rectangle c. Square d. Concentrix circle e.Ellipse f.Line.
Code:

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

```
void main()
{
        Int gd=DETECT,gm,left=100,top=100,right=100,bottom=100,
       x=300,y=150,radius=50;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        rectangle(120,150,230,200);
        circle(x,y,radius);
        // Square
        bar(left+300,top,right+300,bottom);
        line(left-10,top+150,left+410,top+150);
        ellipse(x,y+200,0,360,80,50);
        //Concentric Circle
        for(radius=25;radius<=100;radius=radius+20)
        {
                circle(500,350,radius);
        }
        getch();
        closegraph();
}
```

Output:



PRACTICAL 4

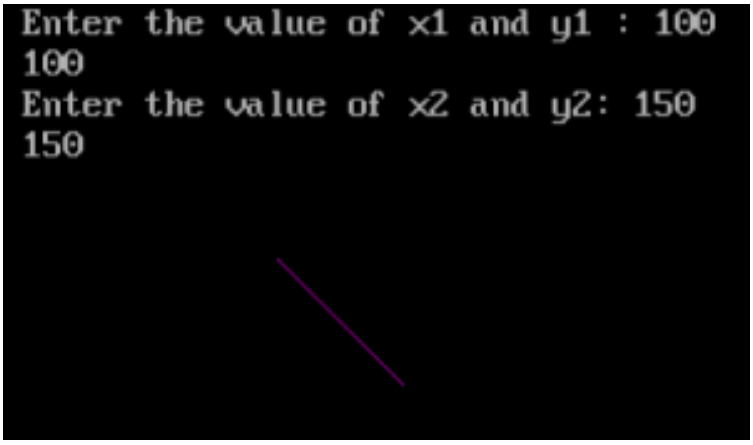
Aim: Write a program to demonstrate DDA line Algorithm.

Code:

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

```
#include<dos.h>
void main()
{
	float m,y,x1,y1,x2,y2,dx,dy,step;
	int i,gd=DETECT,gm;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
	cout<<"Enter the value of x1 and y1:";
	cin>>x1>>y1;
	cout<<"Enter the value of x2 and y2:";
	cin>>x2>>y2;
	dx=abs(x2-x1);
	dy=abs(y2-y1);
	if(dx>=dy)
		step=dx;
	else
		step=dy;
	dx=dx/step;
	dy=dy/step;
	x=x1;
	y=y1;
	i-1;
	while(i<=step)
	{
		putpixel(x,y,5);
		x=x+dx;
		y=y+dy;
		i=i+1;
		delay(50);
	}
	getch();
	closegraph();
}
```

Output:



B

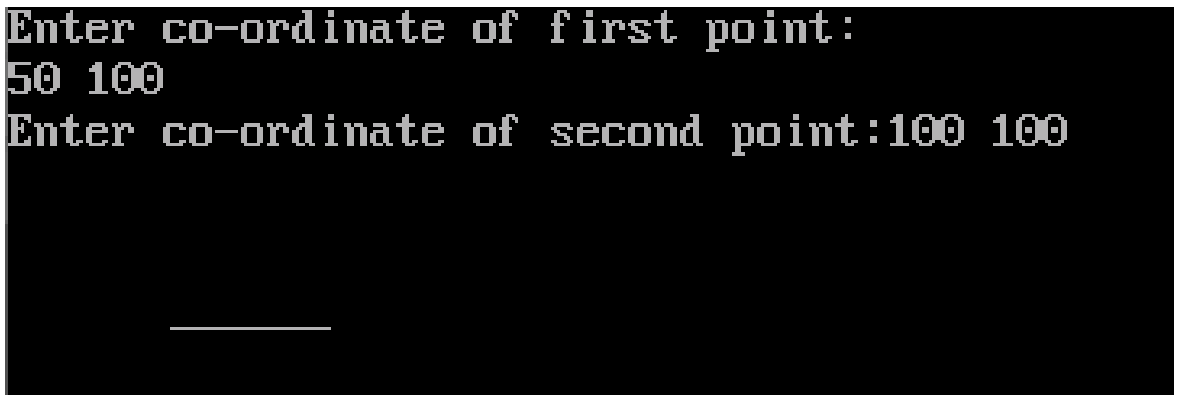
Aim: Program to Demonstrate Breshnam's Line Drawing Algorithm.

Code:

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
void drawline(int x0,int y0,int x1,int y1)
{
        int dx,dy,p,x,y;
        dx=x1-x0;
        dy=y1-y0;
        x=x0;
        y=y0;
        p=2*dy-dx;
        while(x<x1)
```

```
        {
                if(p>=0)
                {
                        putpixel(x,y,7);
                        y=y+1;
                        p=p+2*dy-2*dx;
                }
                else
                {
                        putpixel(x,y,7);
                        p=p+2*dy;
                }
                x=x+1;
        }
}
void main()
{       int gd=DETECT,gm,x0,y0,x1,y1;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        cout<<"Enter coordinate of the first point:";
        cin>>x0>>y0;
        cout<<"Enter coordinate of second point:";
        cin>>x1>>y1;
        drawline(x0,y0,x1,y1);
        getch();
        closegraph();
}
```

Output:



PRACTICAL 5

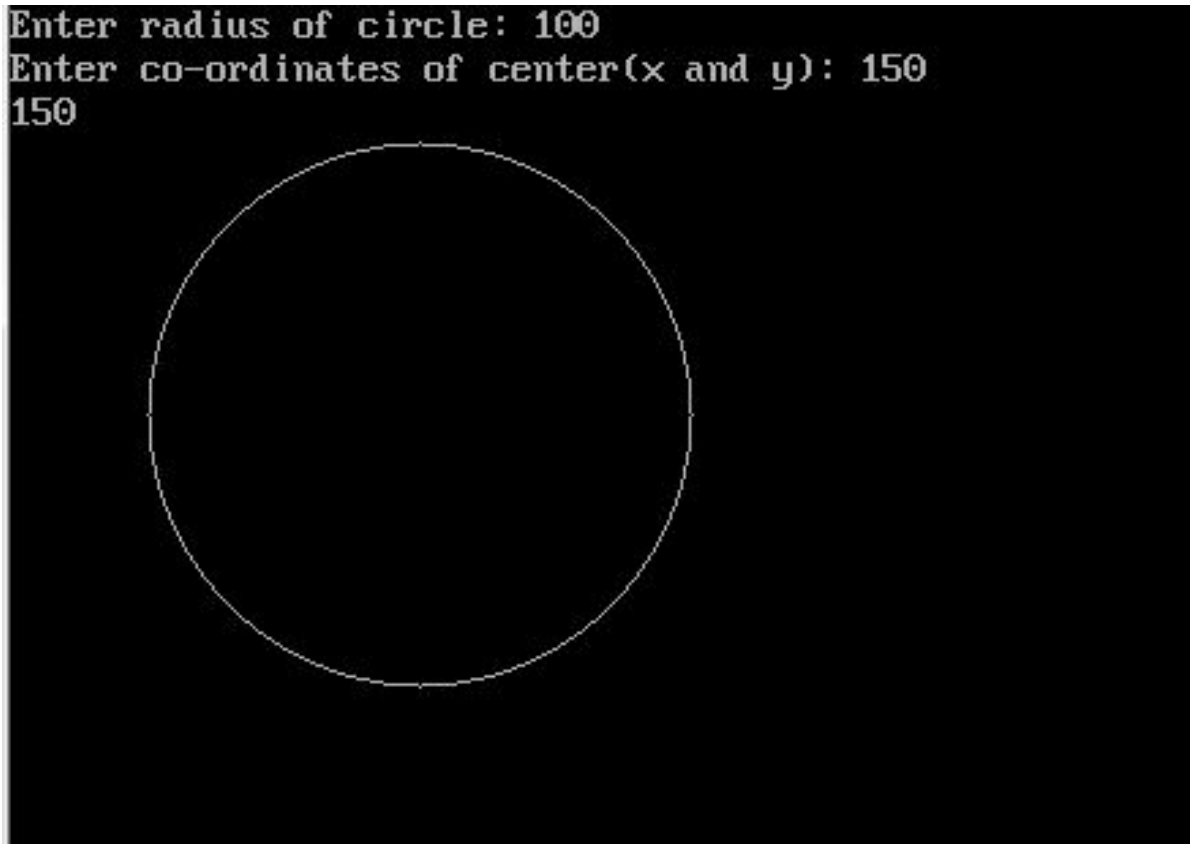
Aim: Program for mid-point circle drawing algorithm.

Code:

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
void drawcircle( int x0,int y0,int radius)
{
	int x=radius;
	int y=0;
	int err=0;
	while(x>=y)
	{
		putpixel(x0+x,y0+y,1);
		putpixel(x0+y,y0+x,2);
		putpixel(x0-y,y0+x,4);
		putpixel(x0-x,y0+y,5);
		putpixel(x0-x,y0-y,9);
		putpixel(x0-y,y0-x,10);
		putpixel(x0+x,y0-y,11);
		putpixel(x0+y,y0-x,14);
		if(err<=0)
		{	y+=1;
			err+=2*y+1;
		}
		if(err>0)
		{	x-=1;
			err-=2*x+1;
		}
		delay(50);
	}
}
void main()
{
	int gd=DETECT,gm,error,x,y,r;
	initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
	cout<<"Enter radius of circle:";
	cin>>r;
	cout<<"Enter coordinates of center (x and y):";
	cin>>x>>y;
	drawcircle(x,y,r);
	getch();
	closegraph();
```

}

Output:


Enter radius of circle: 100
Enter co-ordinates of center(x and y): 150
150

Aim:Develop the program for the mid-point ellipse drawing algorithm.

Code:

```
#include<iostream.h>
#include<dos.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
void display (int xs1, int ys1, int x, int y);
void ellips1(int xs1,int ys1,int rx, int ry)
{
   int x,y;
   float d1,d2,dx,dy;
   x = 0;                // take start position as (0,ry)
   y = ry;               // finding decision parameter d1
   d1 = pow(ry,2) - (pow(rx,2) * ry) + (0.25 * pow(rx,2));
   dx = 2 * pow(ry,2) * x;
   dy = 2 * pow(rx,2) * y;
   do                     // region one
   {
      display(xs1,ys1,x,y);
      if(d1<0)
      {
         x++;
         dx = dx + (2 * (pow(ry,2)));
         d1 = d1 + dx +(pow(ry,2));
      }
      else
      {
         x++;
         y--;
         dx = dx + (2 * (pow(ry,2)));
         dy = dy - (2 * (pow(rx,2)));
         d1 = d1 + dx - dy + (pow(ry,2));
      }
   }while(dx<dy);      // change over condition for region-2
   do                    // region two
   {
      display(xs1,ys1,x,y);
      if(d2>0)
      {
         x = x;
         y--;
         dy = dy - (2 * (pow(rx,2)));
```

```
            d2 = d2 - dy + pow(rx,2);
        }
        else
        {
            x++;
            y--;
            dy = dy - (2 * (pow(rx,2)));
            dx = dx + (2 * (pow(ry,2)));
            d2 = d2 +dx - dy + pow(rx,2);

        }
    }while(y>0);
}
void display(int xs,int ys,int x,int y)
{
    putpixel(xs+x,ys+y,WHITE); // plot points by using 4 point symmetry
    putpixel(xs-x,ys-y,WHITE);
    putpixel(xs+x,ys-y,WHITE);
    putpixel(xs-x,ys+y,WHITE);
}
int main(void)
{
    int xs1,ys1;
    float rx1,ry1;
    int gd = DETECT,gm;          // Initialise the graphics system
    initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
    cout<<"\t\tMidpoint Ellipe Drawing Algorithm\n";
    cout<<"Enter the Center Co-ordinates\n";
    cout<<"xc = \t";
    cin>>xs1;
    cout<<"yc = \t";
    cin>>ys1;
    cout<<"Enter the X Radius\t";
    cin>>rx1;
    cout<<"Enter the Y Radius\t";
    cin>>ry1;
    ellips1(xs1,ys1,rx1,ry1);
    getch();
    closegraph();
}
```

Output:

PRACTICAL 6

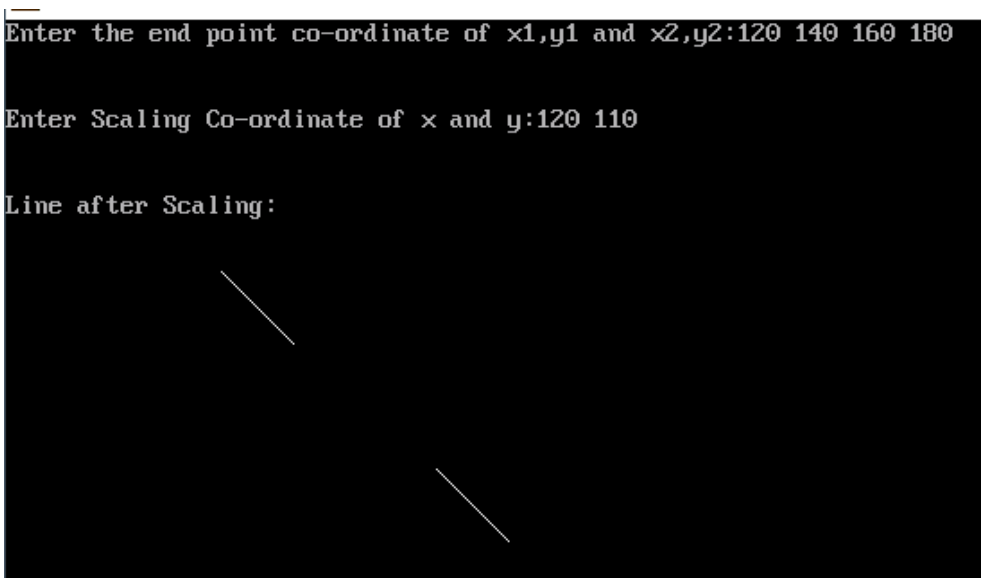
Aim: Program to implement 2D scaling.

Code:

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

```
#include<graphics.h>
#include<math.h>
void main()
{
int i;
int x1,y1,x2,y2,x,y;
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
cout<<"Enter the end point co-ordinate of x1,y1 and x2,y2:";
cin>>x1>>y1>>x2>>y2;
line(x1,y1,x2,y2);
cout<<"\n\nEnter Scaling Co-ordinate of x and y:";
cin>>x>>y;
cout<<"\n\nLine after Scaling:";
x1=(x1+x);
y1=(y1+y);
x2=(x2+x);
y2=(y2+y);
line(x1,y1,x2,y2);
getch();
closegraph();
}
```

Output:



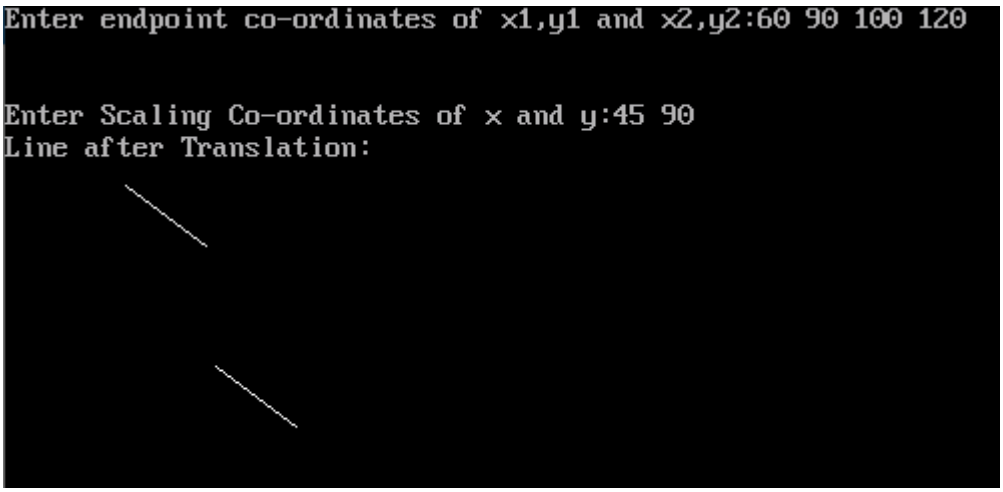
Aim: Program to perform 2D transition.

Code:

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

```
void main()
{
int i;
int x1,y1,x2,y2,x,y;
int gd=DETECT,gm;
initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
cout<<"Enter endpoint co-ordinates of x1,y1 and x2,y2:";
cin>>x1>>y1>>x2>>y2;
line(x1,y1,x2,y2);
cout<<"\n\n\Enter Scaling Co-ordinates of x and y:";
cin>>x>>y;
cout<<"Line after Translation:";
x1=x1+x;
y1=y1+y;
x2=x2+x;
y2=y2+y;
line(x1,y1,x2,y2);
getch();
closegraph();
}
```

Output:



PRACTICALS 7

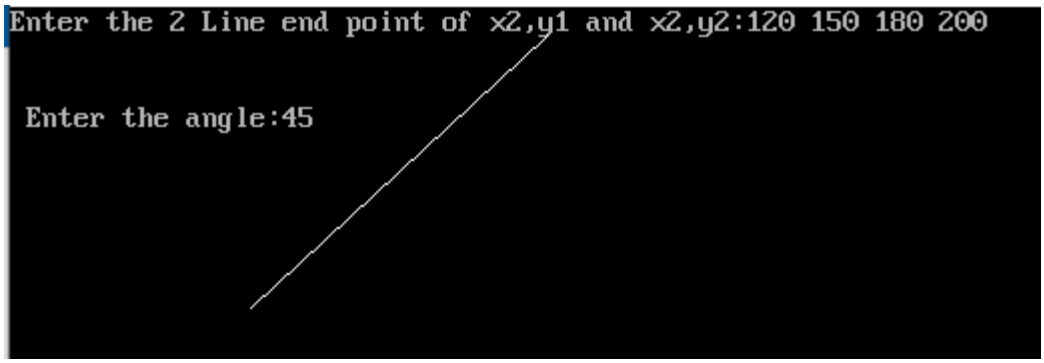
Aim: Program to perform 2D rotation of a given object.

Code:

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>
void main()
{
```

```
        int  gd=DETECT,gm;
        int x1,y1,x2,y2,xn,yn;
        double r11,r12,r21,r22,th;
        initgraph(&gd,&gm,"C:\\TURBOC3\\BGI");
        cout<<"Enter the 2 Line end point of x2,y1 and x2,y2:");
        cin>>x1>>y1>>x2>>y2;
        cout<<"\n\n Enter the angle:";
        cin>>th;
        r11=cos((th*3.1428)/180);
        r12=sin((th*3.1428)/180);
        r21=(-sin((th*3.1428)/180));
        r22=cos((th*3.1428)/180);
        xn=((x2*r11)-(y2*r21));
        yn=((x2*r21)+(y2*r22));
        line(x1,y1,xn,yn);
        getch();
        closegraph();
}
}
```

Output:



Aim: Program to create a house like figure and perform the following operations.

Code:

i. Scaling about the origin followed by translation.
ii. Scaling with reference to an arbitrary point.
iii. Reflect about the line y = mx + c.

```
#include <stdio.h>
#include <graphics.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
void reset (int h[][2])
```

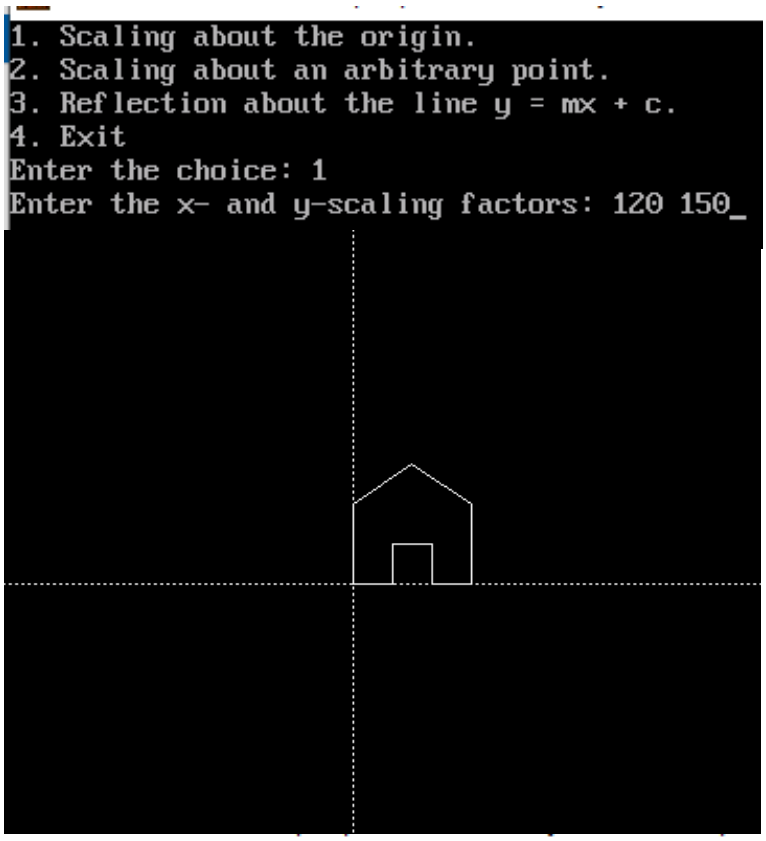
```
{
   int val[9][2] = {
                              { 50, 50 },{ 75, 50 },{ 75, 75 },{ 100, 75 },
                              { 100, 50 },{ 125, 50 },{ 125, 100 },{ 87, 125 },{ 50, 100 }
                        };
   int i;
   for (i=0; i<9; i++)
   {
         h[i][0] = val[i][0]-50;
         h[i][1] = val[i][1]-50;
   }
}
void draw (int h[][2])
{
   int i;
   setlinestyle (DOTTED_LINE, 0, 1);
   line (320, 0, 320, 480);
   line (0, 240, 640, 240);
   setlinestyle (SOLID_LINE, 0, 1);
   for (i=0; i<8; i++)
         line (320+h[i][0], 240-h[i][1], 320+h[i+1][0], 240-h[i+1][1]);
   line (320+h[0][0], 240-h[0][1], 320+h[8][0], 240-h[8][1]);
}
void rotate (int h[][2], float angle)
{
   int i;
   for (i=0; i<9; i++)
   {
         int xnew, ynew;
         xnew = h[i][0] * cos (angle) - h[i][1] * sin (angle);
         ynew = h[i][0] * sin (angle) + h[i][1] * cos (angle);
         h[i][0] = xnew; h[i][1] = ynew;
   }
}
void scale (int h[][2], int sx, int sy)
{
   int i;
   for (i=0; i<9; i++)
   {
         h[i][0] *= sx;
         h[i][1] *= sy;
   }
}
void translate (int h[][2], int dx, int dy)
```

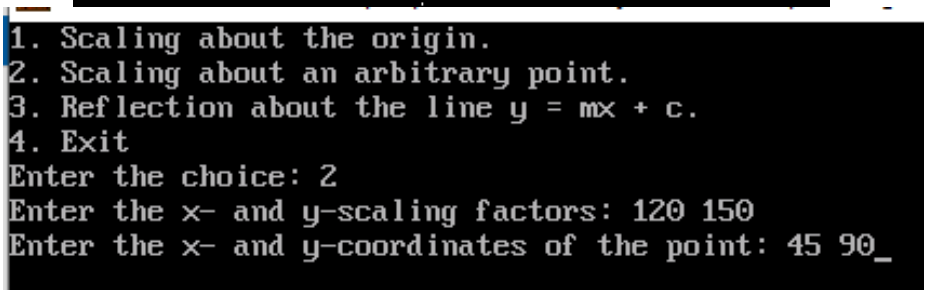
```
{
   int i;
   for (i=0; i<9; i++)
   {
         h[i][0] += dx;
         h[i][1] += dy;
   }
}
void reflect (int h[][2], int m, int c)
{
         int i;
         float angle;
         for (i=0; i<9; i++)
                  h[i][1] -= c;
         angle = M_PI/2 - atan (m);
         rotate (h, angle);
         for (i=0; i<9; i++)
                  h[i][0] = -h[i][0];
         angle = -angle;
         rotate (h, angle);
         for (i=0; i<9; i++)
                  h[i][1] += c;
}
void ini()
{
         int gd=DETECT,gm;
         initgraph(&gd,&gm,"..\\bgi");
}
void dini()
{
         getch();
         closegraph();
}
void main()
{
  int h[9][2],sx,sy,x,y,m,c,choice;
         do
         {
                  clrscr();
                  printf("1. Scaling about the origin.\n");
                  printf("2. Scaling about an arbitrary point.\n");
                  printf("3. Reflection about the line y = mx + c.\n");
                  printf("4. Exit\n");
                  printf("Enter the choice: ");
```

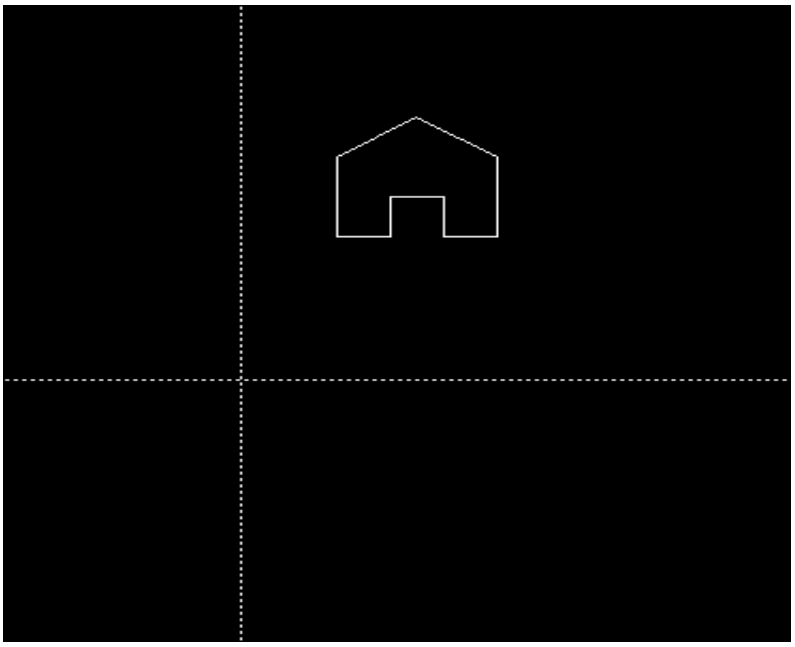
```
        scanf("%d",&choice);
        switch(choice)
        {
                case 1: printf ("Enter the x- and y-scaling factors: ");
                        scanf ("%d%d", &sx, &sy);
                        ini();
                        reset (h);
                        draw (h);getch();
                        scale (h, sx, sy);
                        cleardevice();
                        draw (h);
                        dini();
                        break;
case 2: printf ("Enter the x- and y-scaling factors: ");
                scanf ("%d%d", &sx, &sy);
                printf ("Enter the x- and y-coordinates of the point: ");
                scanf ("%d%d", &x, &y);
                ini();
                reset (h);
                translate (h, x, y);// Go to arbitrary point
                        draw(h); getch();//Show its arbitrary position
                        cleardevice();
                        translate(h,-x,-y);//Take it back to origin
                        draw(h);
                        getch();
                        cleardevice();
                        scale (h, sx, sy);//Now Scale it
                        draw(h);
                        getch();
                        translate (h, x, y);//Back to Arbitrary point
                        cleardevice();
                        draw (h);
                        putpixel (320+x, 240-y, WHITE);
                        dini();
                        break;
                case 3: printf ("Enter the values of m and c: ");
                        scanf ("%d%d", &m, &c);
                        ini();
                        reset (h);
                        draw (h); getch();
                        reflect (h, m, c);
                        cleardevice();
                        draw (h);
                        dini();
```

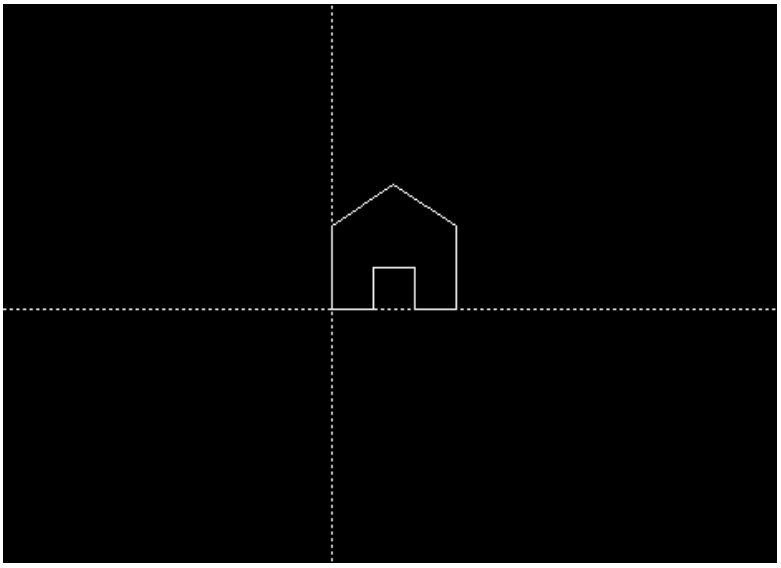
```
                    break;
                case 4: exit(0);
            }
        }while(choice!=4);
}
```

Output:

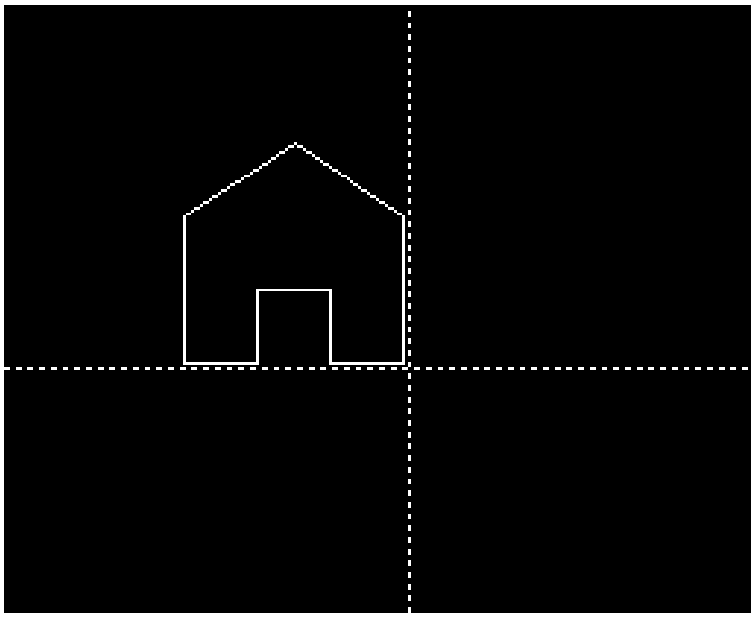


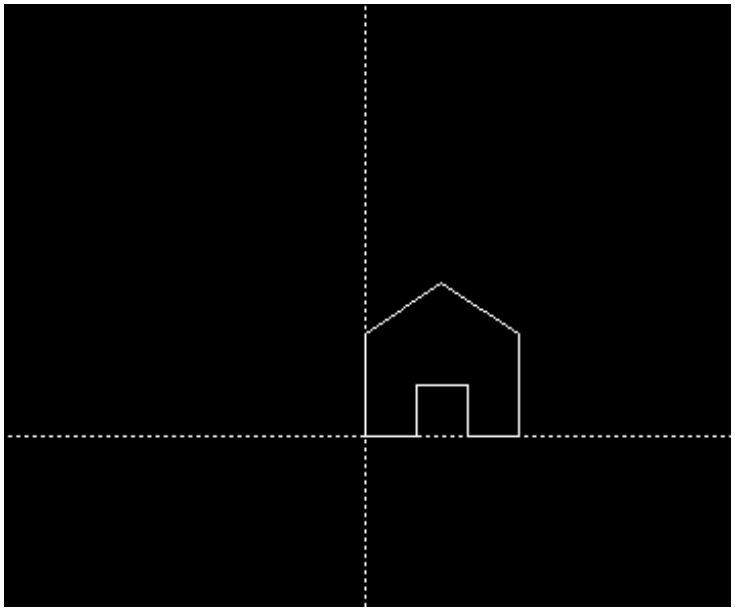






```
1. Scaling about the origin.
2. Scaling about an arbitrary point.
3. Reflection about the line y = mx + c.
4. Exit
Enter the choice: 3
Enter the values of m and c: 120 200
```





PRACTICAL 8

Aim: Write a program to implement Cohen-Sutherland clipping

Code:

```
#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<graphics.h>
#include<dos.h>
typedef struct coordinate
{
        int x,y;
```

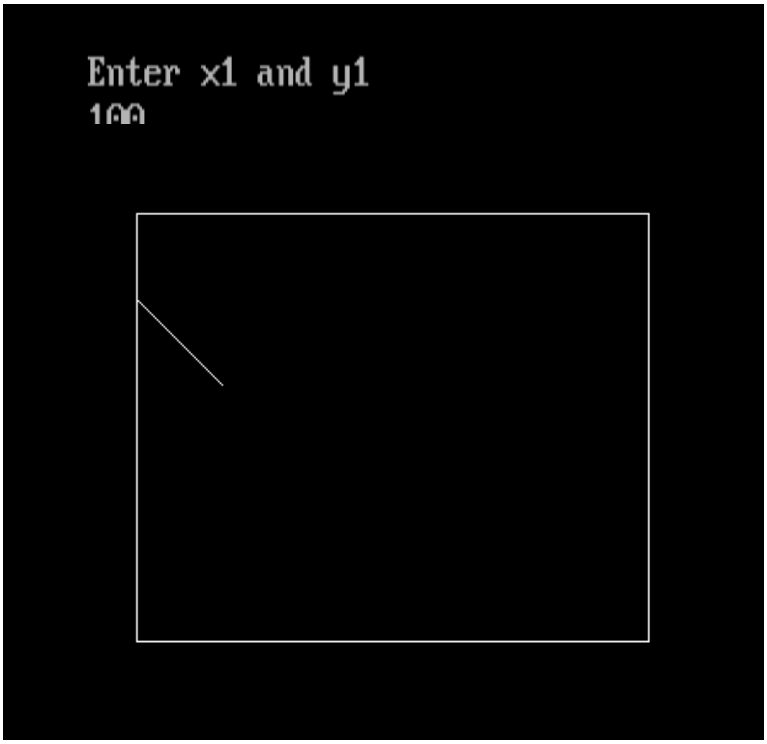
```
        char code[4];
}PT;
void drawwindow();
void drawline(PT p1,PT p2);
PT setcode(PT p);
int visibility(PT p1,PT p2);
PT resetendpt(PT p1,PT p2);
void main()
{
        int gd=DETECT,v,gm;
        PT p1,p2,p3,p4,ptemp;
        printf("\nEnter x1 and y1\n");
        scanf("%d %d",&p1.x,&p1.y);
        printf("\nEnter x2 and y2\n");
        scanf("%d %d",&p2.x,&p2.y);
        initgraph(&gd,&gm,"c:\\turboc3\\bgi");
        drawwindow();
        delay(500);
        drawline(p1,p2);
        delay(500);
        cleardevice();
        delay(500);
        p1=setcode(p1);
        p2=setcode(p2);
        v=visibility(p1,p2);
        delay(500);
        switch(v)
        {
        case 0: drawwindow();
                        delay(500);
                        drawline(p1,p2);
                        break;
        case 1: drawwindow();
                        delay(500);
                        break;
        case 2: p3=resetendpt(p1,p2);
                        p4=resetendpt(p2,p1);
                        drawwindow();
                        delay(500);
                        drawline(p3,p4);
                        break;
        }
        delay(5000);
        closegraph();
```

```
}
 void drawwindow()
{
	line(150,100,450,100);
	line(450,100,450,350);
	line(450,350,150,350);
	line(150,350,150,100);
}
void drawline(PT p1,PT p2)
{
	line(p1.x,p1.y,p2.x,p2.y);
}

PT setcode(PT p)
{
	PT ptemp;

	if(p.y<100)
		ptemp.code[0]='1';
	else
		ptemp.code[0]='0';

	if(p.y>350)
		ptemp.code[1]='1';
	else
		ptemp.code[1]='0';

	if(p.x>450)
		ptemp.code[2]='1';
	else
		ptemp.code[2]='0';

	if(p.x<150)
		ptemp.code[3]='1';
	else
		ptemp.code[3]='0';
	ptemp.x=p.x;
	ptemp.y=p.y;
	return(ptemp);
}

int visibility(PT p1,PT p2)
{
	int i,flag=0;
```

```
	for(i=0;i<4;i++)
	{
		if((p1.code[i]!='0') || (p2.code[i]!='0'))
			flag=1;
	}

	if(flag==0)
		return(0);

	for(i=0;i<4;i++)
	{
		if((p1.code[i]==p2.code[i]) && (p1.code[i]=='1'))
			flag='0';
	}
	if(flag==0)
	return(1);
	return(2);
}

PT resetendpt(PT p1,PT p2)
{
	PT temp;
	int x,y,i;
	float m,k;
	if(p1.code[3]=='1')
		x=150;
	if(p1.code[2]=='1')
		x=450;
	if((p1.code[3]=='1') || (p1.code[2]=='1'))
	{
		m=(float)(p2.y-p1.y)/(p2.x-p1.x);
		k=(p1.y+(m*(x-p1.x)));
		temp.y=k;
		temp.x=x;

		for(i=0;i<4;i++)
			temp.code[i]=p1.code[i];

		if(temp.y<=350 && temp.y>=100)
			return (temp);
	}

	if(p1.code[0]=='1')
```

```
            y=100;
        if(p1.code[1]=='1')
            y=350;
        if((p1.code[0]=='1') || (p1.code[1]=='1'))
        {
            m=(float)(p2.y-p1.y)/(p2.x-p1.x);
            k=(float)p1.x+(float)(y-p1.y)/m;
            temp.x=k;
            temp.y=y;

            for(i=0;i<4;i++)
                temp.code[i]=p1.code[i];
            return(temp);
        }
        else
            return(p1);
}
```

Output:



Aim: Write a program to implement Liang - Barsky Line Clipping Algorithm

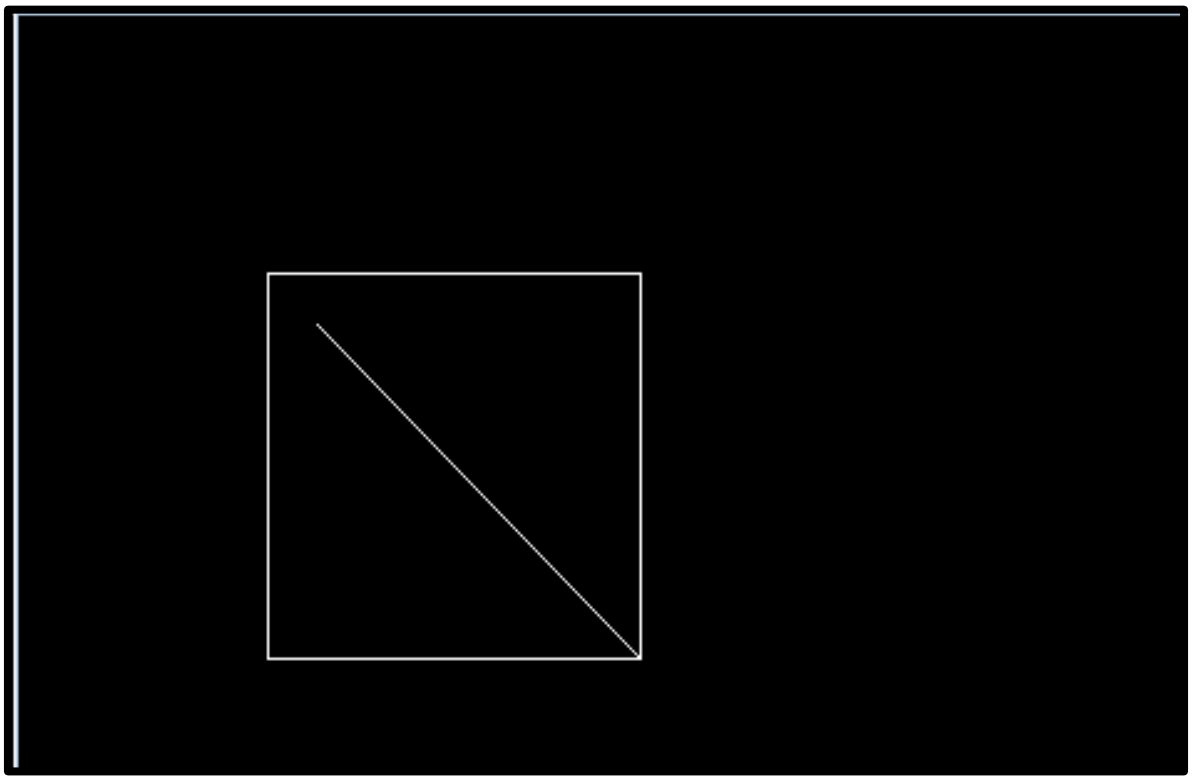
Code:

```
#include<stdio.h>
#include<graphics.h>
#include<math.h>
#include<dos.h>
void main()
{
	int i,gd=DETECT,gm;
	int x1,y1,x2,y2,xmin,xmax,ymin,ymax,xx1,xx2,yy1,yy2,dx,dy;
	float t1,t2,p[4],q[4],temp;
	x1=120;
	y1=120;
	x2=300;
	y2=300;
	xmin=100;
	ymin=100;
	xmax=250;
	ymax=250;
	initgraph(&gd,&gm,"c:\\turboc3\\bgi");
	rectangle(xmin,ymin,xmax,ymax);
	dx=x2-x1;
	dy=y2-y1;

	p[0]=-dx;
	p[1]=dx;
	p[2]=-dy;
	p[3]=dy;
	q[0]=x1-xmin;
	q[1]=xmax-x1;
	q[2]=y1-ymin;
	q[3]=ymax-y1;

	for(i=0;i<4;i++)
	{
		if(p[i]==0)
		{
			printf("line is parallel to one of the clipping boundary");
			if(q[i]>=0)
			{
				if(i<2)
```

```
                        {
                                if(y1<ymin)
                                {
                                        y1=ymin;
                                }

                                if(y2>ymax)
                                {
                                        y2=ymax;
                                }

                                line(x1,y1,x2,y2);
                        }

                        if(i>1)
                        {
                                if(x1<xmin)
                                {
                                        x1=xmin;
                                }

                                if(x2>xmax)
                {
                  x2=xmax;
                }

        ine(x1,y1,x2,y2);
}
}
}
}
t1=0;
t2=1;
for(i=0;i<4;i++)
{
        temp=q[i]/p[i];
        if(p[i]<0)
        {
                if(t1<=temp)
                t1=temp;
        }
        else
        {
                if(t2>temp)
```

```
                t2=temp;
            }
        }
        if(t1<t2)
        {
            xx1 = x1 + t1 * p[1];
            xx2 = x1 + t2 * p[1];
            yy1 = y1 + t1 * p[3];
            yy2 = y1 + t2 * p[3];
            line(xx1,yy1,xx2,yy2);
        }
        delay(5000);
        closegraph();
}
```

Output:



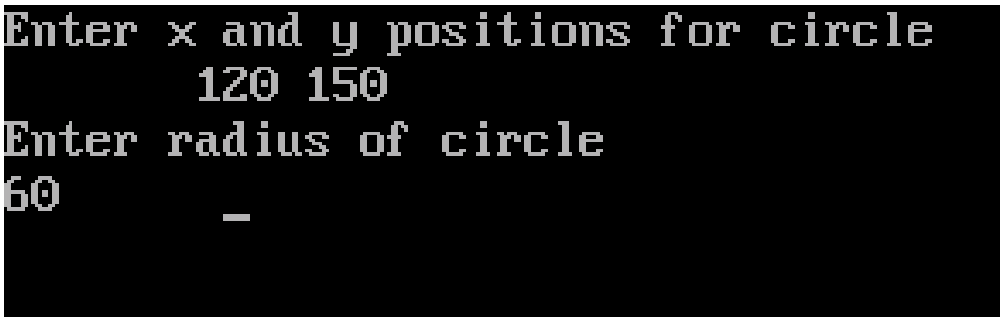
PRACTICAL 9

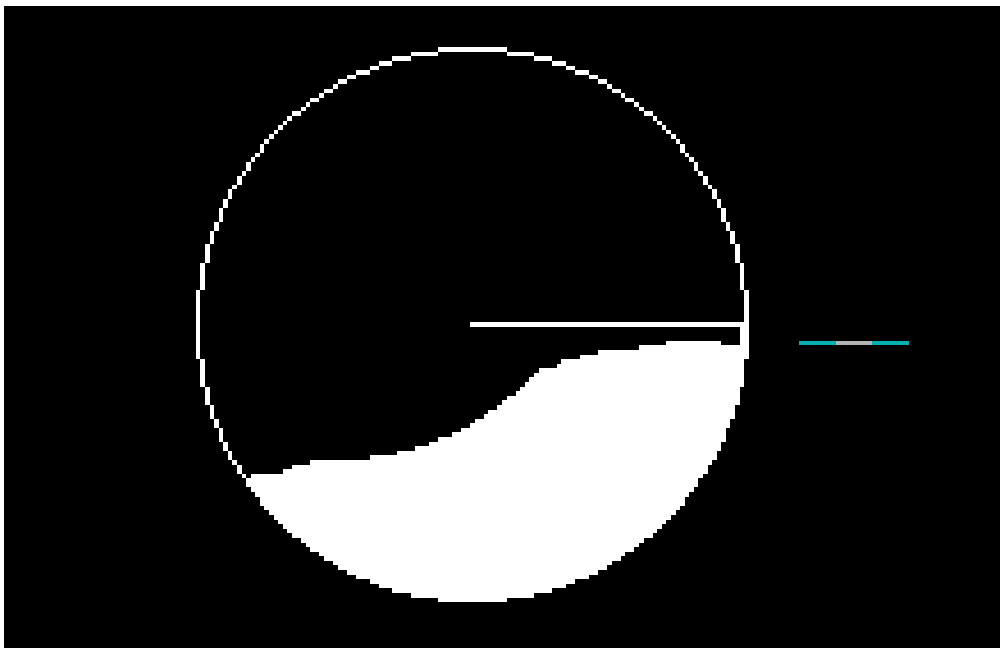
9.A Write a program to fill a circle using Flood Fill Algorithm.

Code:

```
#include<iostream.h>
#include<graphics.h>
#include<dos.h>
void floodFill(int x,int y,int oldcolor,int newcolor)
{
        if(getpixel(x,y) == oldcolor)
        {       putpixel(x,y,newcolor);
                floodFill(x+1,y,oldcolor,newcolor);
                floodFill(x,y+1,oldcolor,newcolor);
                floodFill(x-1,y,oldcolor,newcolor);
                floodFill(x,y-1,oldcolor,newcolor);
        }
}
int main()
{
        int gm,gd=DETECT,radius;
        int x,y;
        cout<<"Enter x and y positions for circle\n";
        cin>>x>>y;
        cout<<"Enter radius of circle\n";
        cin>>radius;
        initgraph(&gd,&gm,"c:\\turboc3\\bgi");
        circle(x,y,radius);
        floodFill(x,y,0,15);
        delay(50);
        closegraph();
       return 0;
}
```

Output:





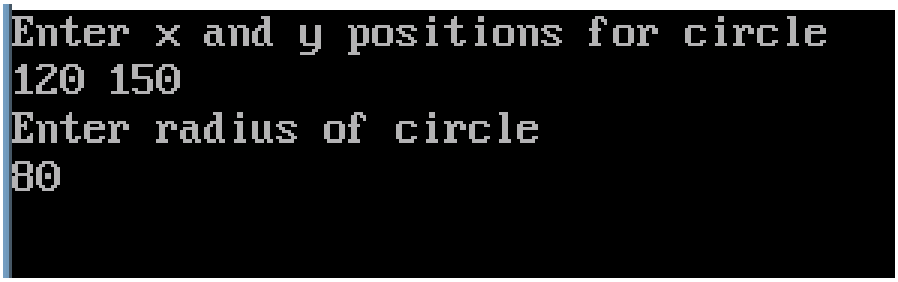
PRACTICAL 10

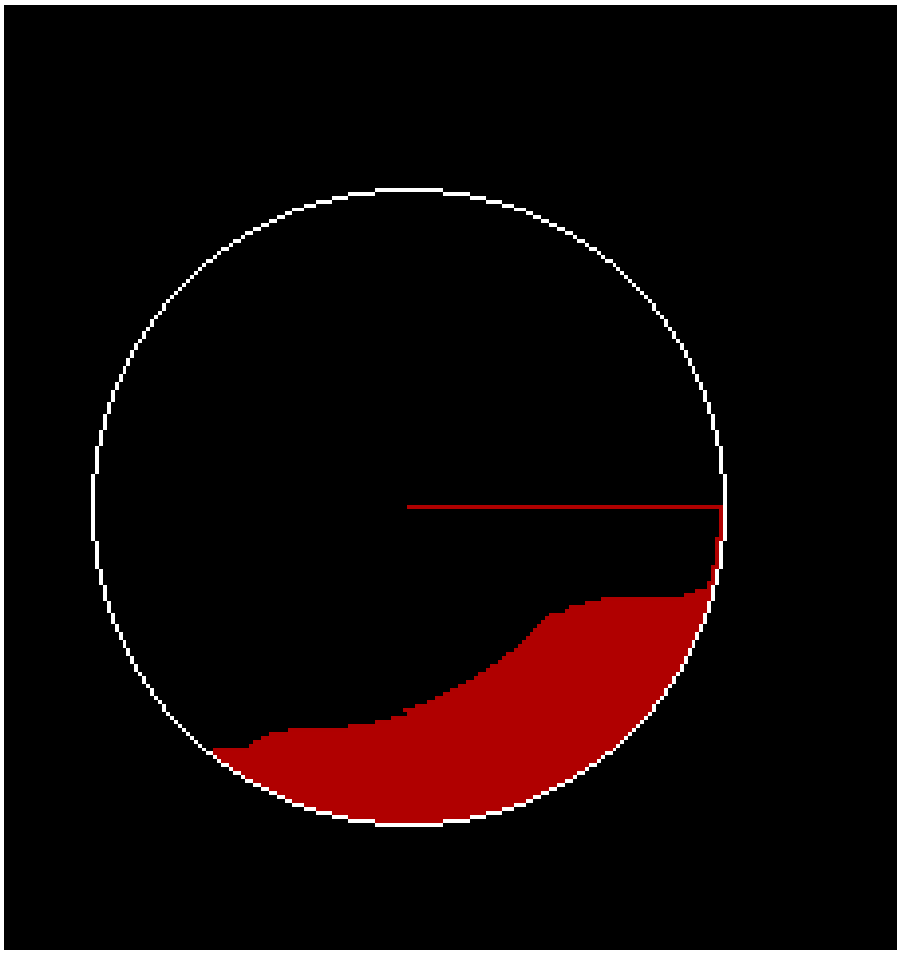
Aim: Write a program to fill a circle using Boundary Fill Algorithm.

Code:

```
#include<iostream.h>
#include<graphics.h>
#include<dos.h>
void boundaryfill(int x,int y,int f_color,int b_color)
{       if(getpixel(x,y)!=b_color && getpixel(x,y)!=f_color)
        {       putpixel(x,y,f_color);
                boundaryfill(x+1,y,f_color,b_color);
                boundaryfill(x,y+1,f_color,b_color);
                boundaryfill(x-1,y,f_color,b_color);
                boundaryfill(x,y-1,f_color,b_color);
        }
}
int main()
{
        int gm,gd=DETECT,radius;
        int x,y;
        cout<<"Enter x and y positions for circle\n";
        cin>>x>>y;
        cout<<"Enter radius of circle\n";
        cin>>radius;
        initgraph(&gd,&gm,"c:\\turboc3\\bgi");
        circle(x,y,radius);
        boundaryfill(x,y,4,15);
        delay(50);
        closegraph();
        return 0;
}
```

Output:





Aim: Perform smiling face animation using graphic functions.

Code:

#include<iostream.h>

```
#include <graphics.h>
#include <conio.h>
int main()
{       int gd=DETECT,gm;
        initgraph(&gdr,&gm,"C:\\TURBOC3\\BGI");
        outtextxy(10, 10 + 10, "Program to draw a smiley face in C graphics");
        setcolor(YELLOW);
        circle(300, 100, 40);
        setfillstyle(SOLID_FILL, YELLOW);
        floodfill(300, 100, YELLOW);
        setcolor(BLACK);
        setfillstyle(SOLID_FILL, BLACK);
        fillellipse(310, 85, 2, 6);
        fillellipse(290, 85, 2, 6);
        ellipse(300, 100, 205, 335, 20, 9);
        ellipse(300, 100, 205, 335, 20, 10);
        ellipse(300, 100, 205, 335, 20, 11);
        getch();
        return 0;}
```

Output:

Aim: Draw the moving car on the screen.

Code:

```
#include <stdio.h>
#include <graphics.h>
#include <conio.h>
#include <dos.h>
void main()
{
   int gd = DETECT, gm;
   int i, maxx, midy;
   initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");
   maxx = getmaxx();
   midy = getmaxy()/2;
   for (i=0; i < maxx-150; i=i+5) {
      cleardevice();
      setcolor(WHITE);
      line(0, midy + 37, maxx, midy + 37);
      setcolor(YELLOW);
      setfillstyle(SOLID_FILL, RED);
      line(i, midy + 23, i, midy);
      line(i, midy, 40 + i, midy - 20);
      line(40 + i, midy - 20, 80 + i, midy - 20);
      line(80 + i, midy - 20, 100 + i, midy);
      line(100 + i, midy, 120 + i, midy);
      line(120 + i, midy, 120 + i, midy + 23);
      line(0 + i, midy + 23, 18 + i, midy + 23);
      arc(30 + i, midy + 23, 0, 180, 12);
      line(42 + i, midy + 23, 78 + i, midy + 23);

      arc(90 + i, midy + 23, 0, 180, 12);
      line(102 + i, midy + 23, 120 + i, midy + 23);
```

```
        line(28 + i, midy, 43 + i, midy - 15);
        line(43 + i, midy - 15, 57 + i, midy - 15);
        line(57 + i, midy - 15, 57 + i, midy);
        line(57 + i, midy, 28 + i, midy);
        line(62 + i, midy - 15, 77 + i, midy - 15);
        line(77 + i, midy - 15, 92 + i, midy);
        line(92 + i, midy, 62 + i, midy);
        line(62 + i, midy, 62 + i, midy - 15);
        floodfill(5 + i, midy + 22, YELLOW);
        setcolor(BLUE);
        setfillstyle(SOLID_FILL, DARKGRAY);
        circle(30 + i, midy + 25, 9);
        circle(90 + i, midy + 25, 9);
        floodfill(30 + i, midy + 25, BLUE);
        floodfill(90 + i, midy + 25, BLUE);
        delay(100);}
    getch();
    closegraph();
}
```

Output: