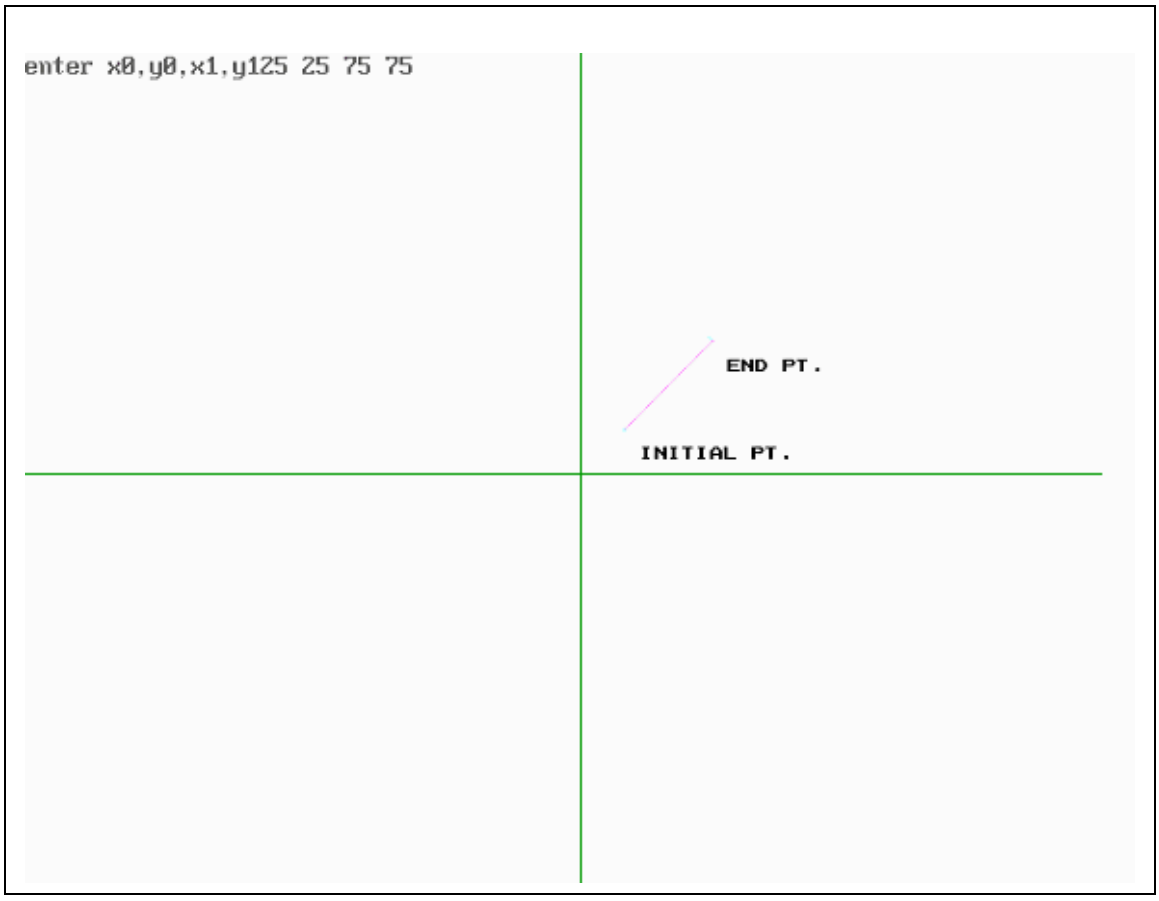
CONTENTS

LINE DRAWING – DDA

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

void put_pixel(int x, int y, int col)
{
 putpixel(x+320, 240-y, col);
}

int round(float x)
{
 double rem = fmod((double)x,1.0);
 if(x<0.5)
   return (floor((double)x));
 else
   return (ceil((double)x));
}


void dda(int x1, int y1, int x2, int y2)
{
 int xa,ya,xb,yb;
 setcolor(RED);
 line(320,0,320,480);
 setcolor(BLUE);
 line(0,240,640,240);
 setcolor(WHITE);

 if(x1<x2)
 {
  xa=x1;ya=y1;
  xb=x2;yb=y2;
 }
 else
 {
  xa=x2;ya=y2;
  xb=x1;yb=y1;
 }

 int dx,dy;
 dx=xb-xa;
 dy=yb-ya;
 int steps;
 float x=xa,y=ya;
 if (abs(dx)>abs(dy))
   steps = abs(dx);
 else
   steps = abs(dy);
 float xinc,yinc;
 xinc = 1.0*dx/steps;
 yinc = 1.0*dy/steps;
 put_pixel(xa,ya,15);
```

```
  while(x<xb)
  {
    x+=xinc;
    y+=yinc;
    put_pixel(round(x),round(y),15);
  }
}

void main()
{
  clrscr();
  int x1,y1,x2,y2;
  cout<<"Enter x1,y1 : ";
  cin>>x1>>y1;
  cout<<"Enter x2,y2 : ";
  cin>>x2>>y2;

  int gd = DETECT, gm;
  initgraph(&gd,&gm,"c:\\tc\\bgi");
  dda(x1,y1,x2,y2);
  getch();
  closegraph();
}
```

OUTPUT



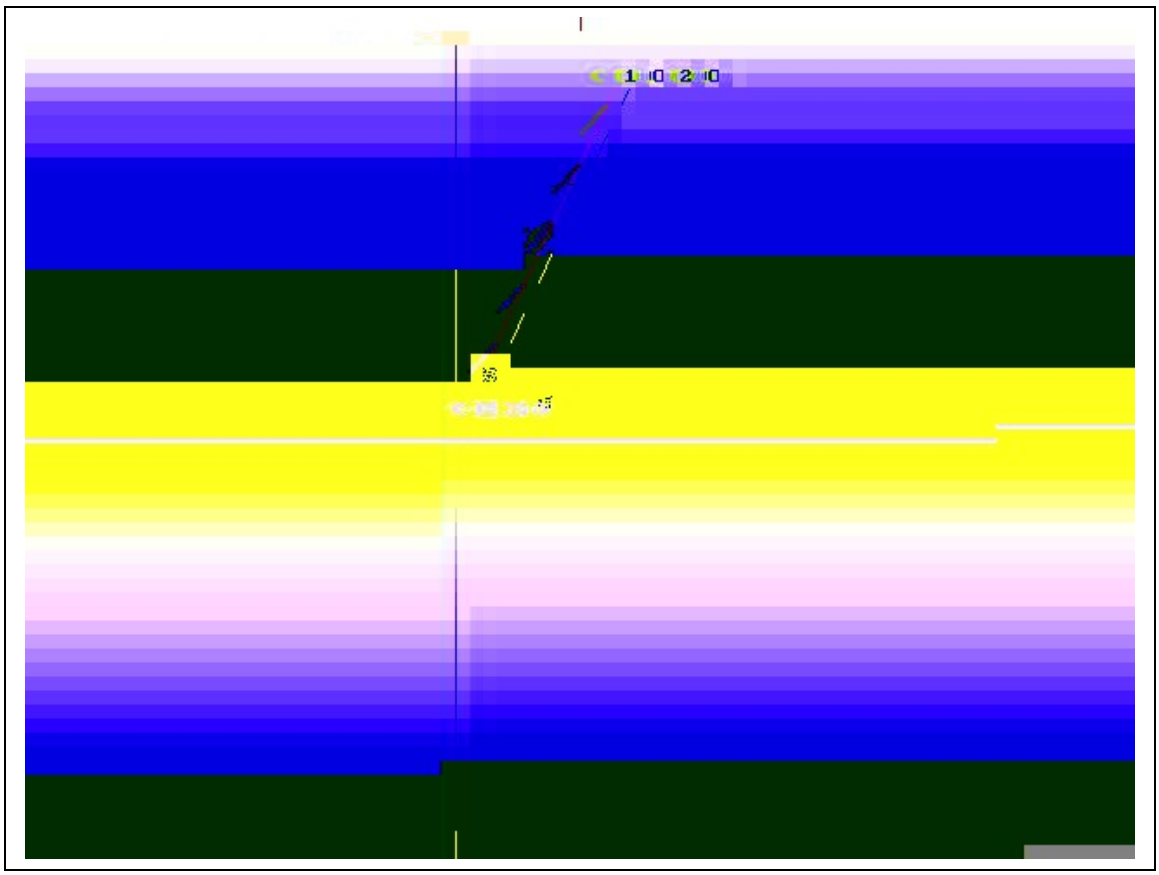
LINE DRAWING - MID POINT

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

void put_pixel(int x, int y, int col)
{
 putpixel(x+320, 240-y, col);
}

void mid_pt(int x1, int y1, int x2, int y2)
{
 int d;
 float m;
 setcolor(RED);
 line(320,0,320,480);
 setcolor(BLUE);
 line(0,240,640,240);
 setcolor(WHITE);

 if(x2!=x1)
  m=1.0*(y2-y1)/(x2-x1);

 int xa,ya,xb,yb;

 if(x1<x2)
 {
  xa=x1;ya=y1;
  xb=x2;yb=y2;
 }
 else
 {
  xa=x2;ya=y2;
  xb=x1;yb=y1;
 }

 int dx = xb-xa;
 int dy = yb-ya;

 float x = xa, y = ya;
 if( m>=0 && m<=1 )
 {
  put_pixel(xa,ya,15);
  d = 2.0*dy-dx;
  while(x<xb)
  {
   if(d>0)
   {
        //case ne
        d+=dy-dx;
        x++;
        y++;
   }
```

```
   else
   {
         d+= dy;
         x++;
   }
   put_pixel(x,y,15);
  }
}
else if(m>-1 && m<0)
{
  put_pixel(xa,ya,15);
  d = 2.0*dy+dx;
  while(x<xb)
  {
   if(d>0)
   {
         //case e
         d+=dy;
         x++;

   }
   else
   {
         d+= dy+dx;
         x++;  y--;
   }
   put_pixel(x,y,15);
  }
}
else if(m>1)
{
  put_pixel(xa,ya,15);
  d = dy - 2.0*dx;
  while(x<xb)
  {
   if(d>0)
   {
         //case n
         d-= dx;
         y++;
   }
   else
   {
         d+= dy-dx;
         x++;  y++;
   }
   put_pixel(x,y,15);
  }
}
else if(m<-1)
{
  put_pixel(xa,ya,15);
  d = dy + 2.0*dx;
  while(x<xb)
  {
   if(d>0)
```

```
    {
        //case se
        d+=dy+dx;
        x++;y--;
    }
    else
    {
        d+= dx;
        y--;
    }
    put_pixel(x,y,15);
   }
  }
}

void main()
{
 clrscr();
 int x1,y1,x2,y2;
 cout<<"Enter x1,y1 : ";
 cin>>x1>>y1;
 cout<<"Enter x2,y2 : ";
 cin>>x2>>y2;

 int gd = DETECT, gm;
 initgraph(&gd,&gm,"c:\\tc\\bgi");

 mid_pt(x1,y1,x2,y2);
 getch();
 closegraph();
}
```

OUTPUT



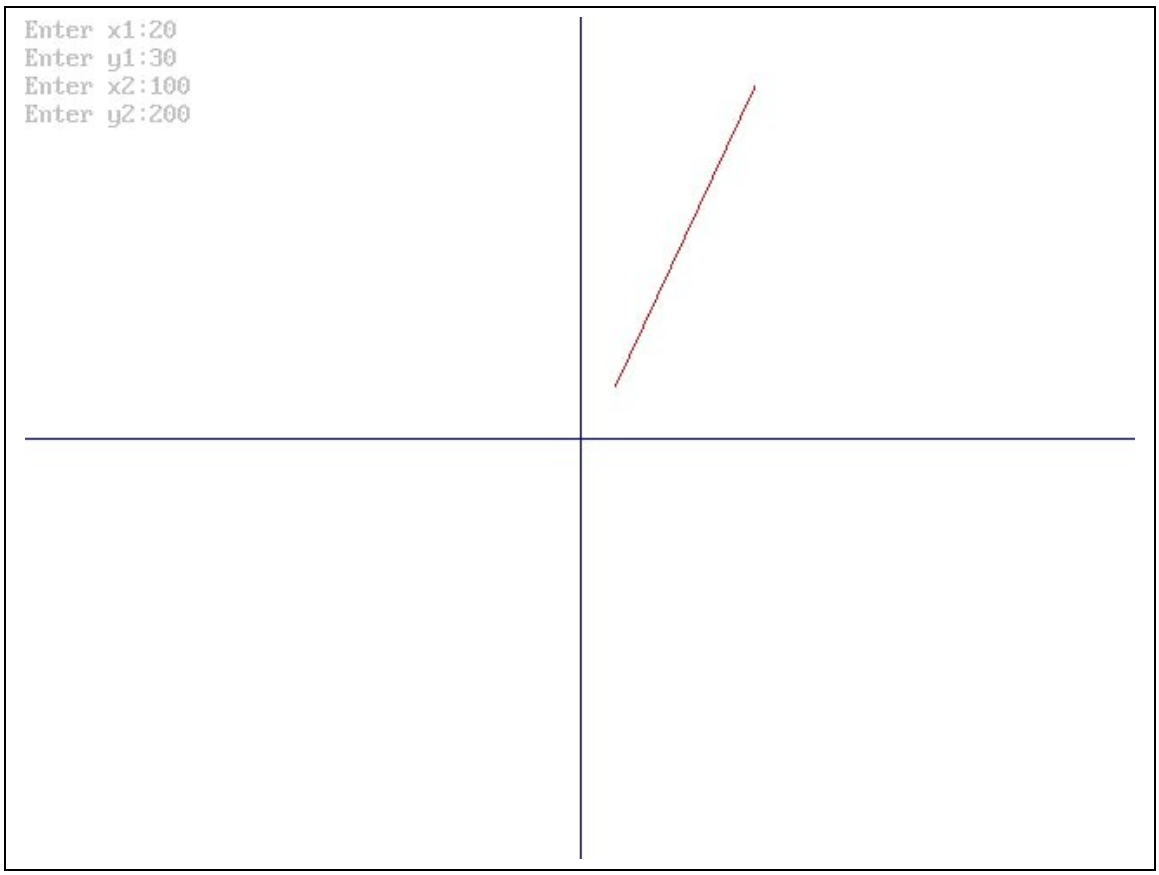
LINE DRAWING – BRESENHAM'S APPROACH

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


void put_pixel(int x, int y, int col)
{
 putpixel(x+320, 240-y, col);
}


void brsnhm_line(int x1, int y1, int x2, int y2)
{

 setcolor(RED);
 line(320,0,320,480);
 setcolor(BLUE);
 line(0,240,640,240);
 setcolor(WHITE);
 int xa, ya,xb,yb;
 if(x1<x2)
 {
  xa=x1;ya=y1;
  xb=x2;yb=y2;
 }
 else
 {
  xa=x2;ya=y2;
  xb=x1;yb=y1;
 }

 int dx,dy;
 dx=xb-xa;
 dy=yb-ya;

 int d;
 float x=xa, y=ya;
 put_pixel(xa,ya,15);

 float m = 1.0*dy/dx;

 if(m>=0 && m<=1)
 {
  d=2*dy-dx;
  while(x<xb)
  {
   if(d<0)
   {
        d+=2*dy;
        x++;
   }
   else
```

```
    {
          d+=2*(dy-dx);
          x++;
          y++;
    }
    put_pixel(x,y,15);
  }
}
else if(m>1)
{
  d=2*dx-dy;
  while(x<xb)
  {
    if(d<0)
    {
          d+=2*dx;
          y++;
    }
    else
    {
          d+=2*(dx-dy);
          x++;
          y++;
    }
    put_pixel(x,y,15);
  }
}
else if(m>=-1 && m<0)
{
  d=-2*dy-dx;
  while(x<xb)
  {
    if(d<0)
    {
          d-=2*dy;
          x++;
    }
    else
    {
          d-=2*(dx+dy);
          y--;
          x++;
    }
    put_pixel(x,y,15);
  }
}
else if(m<-1)
{
  d = -2*dx-dy;
  while(x<xb)
  {
    if(d>0)
    {
          d-= 2*dx;
          y--;
    }
```

```
    else
    {
          d-= 2*(dx+dy);
          y--;
          x++;
    }
    put_pixel(x,y,15);
   }
  }

}

void main()
{
  clrscr();
  int x1,y1,x2,y2;
  cout<<"Enter x1,y1 : ";
  cin>>x1>>y1;
  cout<<"Enter x2,y2 : ";
  cin>>x2>>y2;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

  brsnhm_line(x1,y1,x2,y2);
  getch();
  closegraph();
}
```

OUTPUT



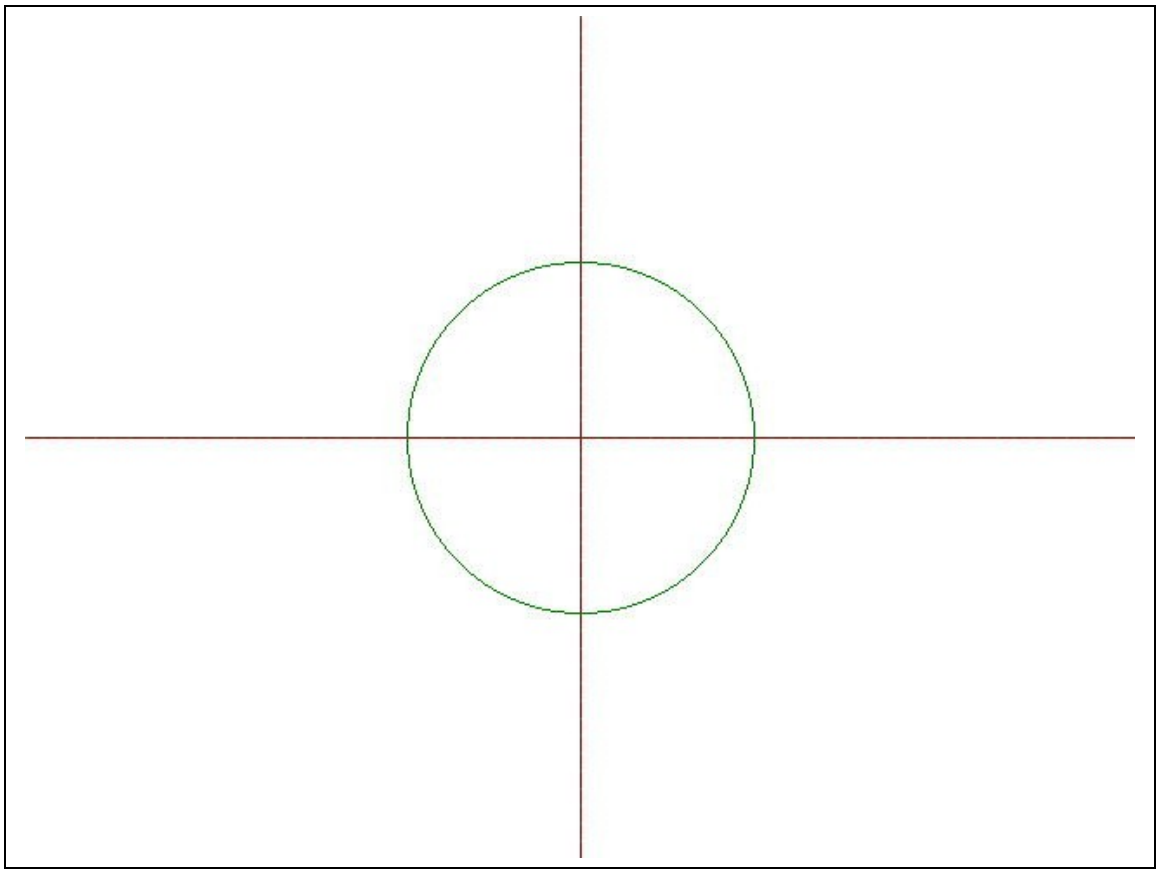
CIRCLE - MID POINT (1st ORDER)

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

void plotpixel(int x, int y,int xc, int yc)
{
 delay(20);
 putpixel((x+xc+320),(240-y-yc),15);
 putpixel((-x+xc+320),(240-y-yc),15);
 putpixel((x+xc+320),(240+y-yc),15);
 putpixel((-x+xc+320),(240+y-yc),15);

 putpixel((y+xc+320),(240-yc-x),15);
 putpixel((-y+xc+320),(240-yc-x),15);
 putpixel((y+xc+320),(240-yc+x),15);
 putpixel((-y+xc+320),(240-yc+x),15);
}

void circ(int r,int xc, int yc)
{
 float x=0,y=r;
 float d = 1-r;
 setcolor(RED);
 line(320,0,320,480);
 setcolor(BLUE);
 line(0,240,640,240);
 setcolor(WHITE);

 plotpixel(x,y,xc,yc);
 while(x<=y)
 {
  if(d<0)
  {
   d+=3+2*x;
   x++;
  }
  else
  {
   d+=5+2*(x-y);
   x++;
   y--;
  }
  plotpixel(x,y,xc,yc);
 }
}

void main()
{
 clrscr();
```

```
  int xc,yc;
  cout<<"Enter co-ordinates of center : ";
  cin>>xc>>yc;


  int r; //radius
  cout<<"Enter the radius : ";
  cin>>r;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

  circ(r,xc,yc);
  getch();
  closegraph();
}
```

OUTPUT

Enter radius : 100

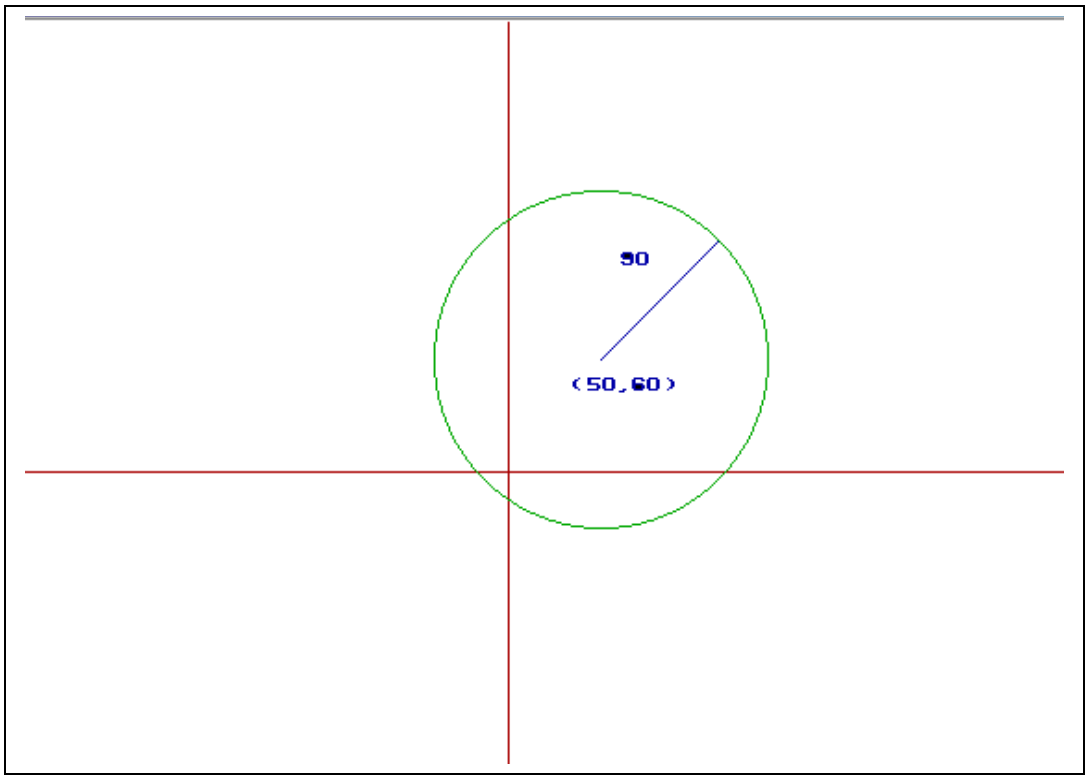


CIRCLE – MID POINT (2nd ORDER)

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

void plotpixel(int x, int y,int xc, int yc)
{
 delay(20);
 putpixel((x+xc+320),(240-y-yc),15);
 putpixel((-x+xc+320),(240-y-yc),15);
 putpixel((x+xc+320),(240+y-yc),15);
 putpixel((-x+xc+320),(240+y-yc),15);

 putpixel((y+xc+320),(240-yc-x),15);
 putpixel((-y+xc+320),(240-yc-x),15);
 putpixel((y+xc+320),(240-yc+x),15);
 putpixel((-y+xc+320),(240-yc+x),15);
}

void circ(int r,int xc, int yc)
{
 float x=0,y=r;
 float d = 1-r;
 float de = 3,dse = 5-2*r;

 setcolor(RED);
 line(320,0,320,480);
 setcolor(BLUE);
 line(0,240,640,240);
 setcolor(WHITE);

 plotpixel(x,y,xc,yc);
 while(x<=y)
 {
  if(d<0)
  {
   d+=de;
   de+=2;
   dse+=2;
   x++;
  }
  else
  {
   d+=dse;
   de+=2;
   dse+=4;
   x++;
   y--;
  }
  plotpixel(x,y,xc,yc);
 }
}
```

```
void main()
{
  clrscr();

  int xc,yc;
  cout<<"Enter co-ordinates of center : ";
  cin>>xc>>yc;

  int r; //radius
  cout<<"Enter the radius : ";
  cin>>r;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

  circ(r,xc,yc);
  getch();
  closegraph();
}
```

OUTPUT

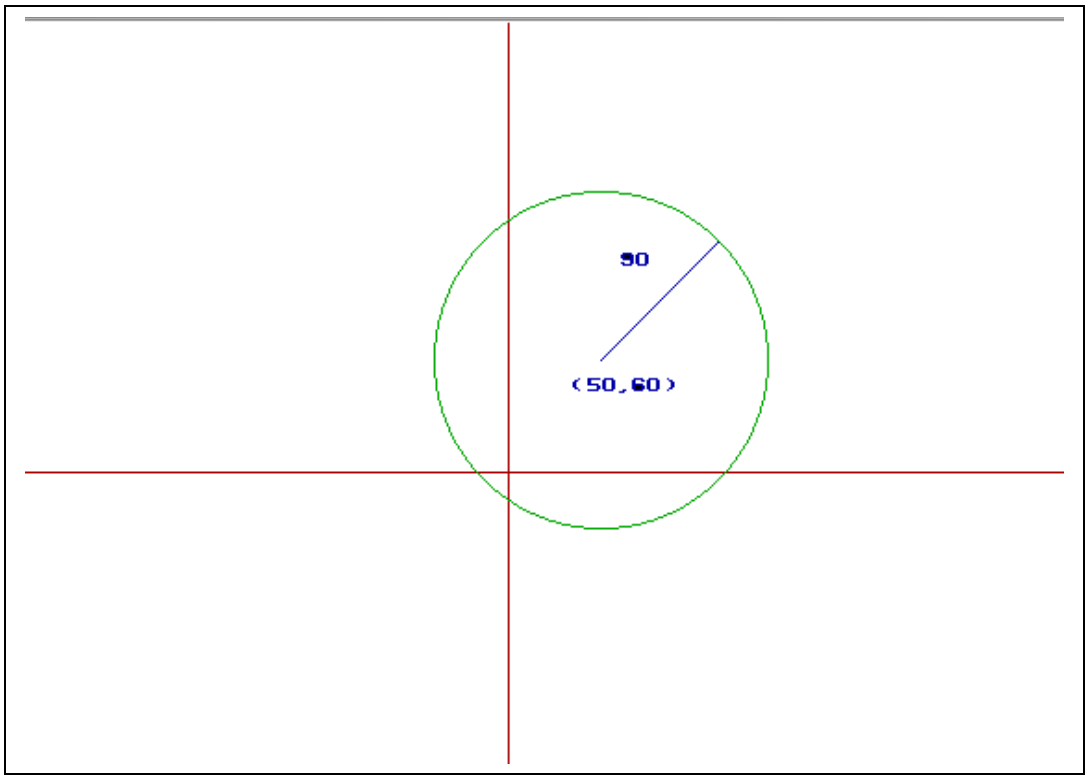


CIRCLE – BRESENHAM'S APPROACH

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

void plotpixel(int x, int y,int xc, int yc)
{
 delay(20);
 putpixel((x+xc+320),(240-y-yc),15);
 putpixel((-x+xc+320),(240-y-yc),15);
 putpixel((x+xc+320),(240+y-yc),15);
 putpixel((-x+xc+320),(240+y-yc),15);

 putpixel((y+xc+320),(240-yc-x),15);
 putpixel((-y+xc+320),(240-yc-x),15);
 putpixel((y+xc+320),(240-yc+x),15);
 putpixel((-y+xc+320),(240-yc+x),15);
}

void circ(int r,int xc,int yc)
{
 int x=0,y=r;
 float d = 3-2*r;

 setcolor(RED);
 line(320,0,320,480);
 setcolor(BLUE);
 line(0,240,640,240);
 setcolor(WHITE);

 plotpixel(x,y,xc,yc);
 while(x<=y)
 {
  if(d<0)
  {
   d+=6+4*x;
   x++;
  }
  else
  {
   d+=10+4*(x-y);
   x++;
   y--;
  }
  plotpixel(x,y,xc,yc);
 }
}

void main()
{
 clrscr();
```

```
  int xc,yc;
  cout<<"Enter co-ordinates of center : ";
  cin>>xc>>yc;

  int r; //radius
  cout<<"Enter the radius : ";
  cin>>r;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

  circ(r,xc,yc);
  getch();
  closegraph();
}
```

OUTPUT

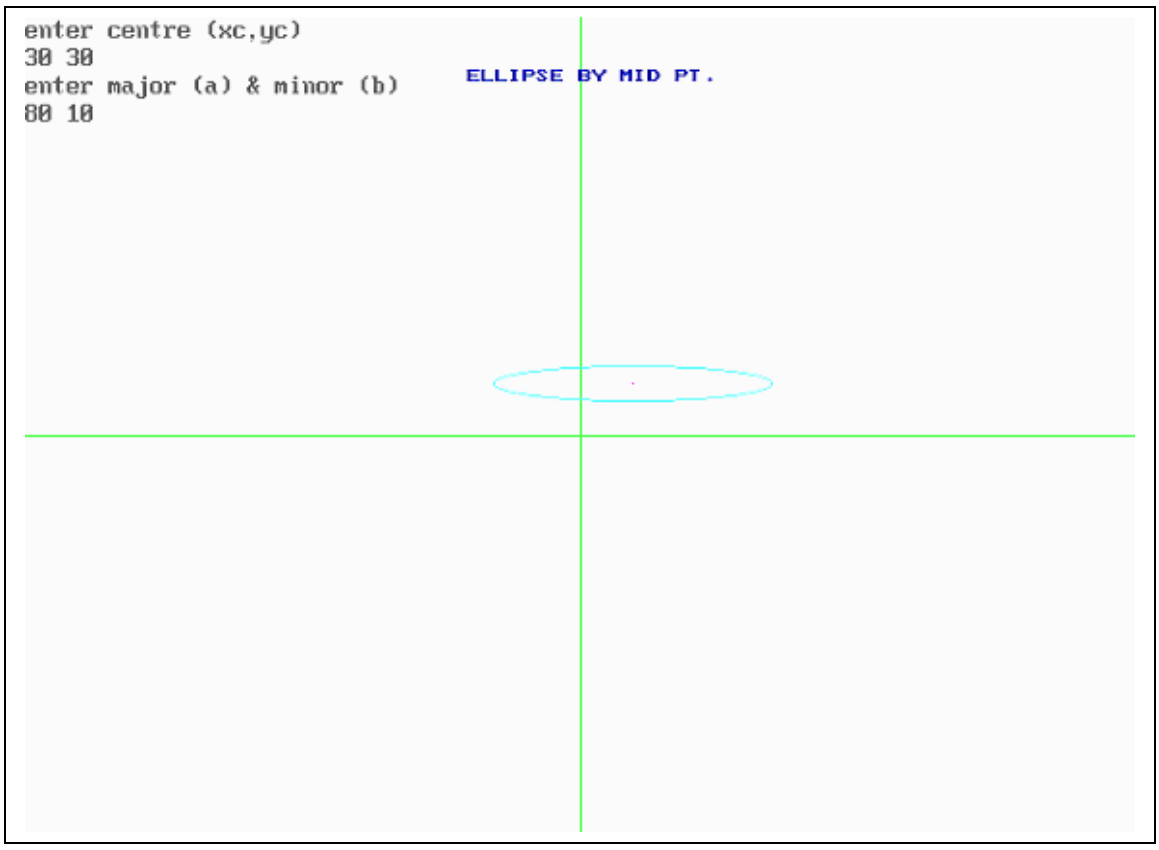


ELLIPSE – MID POINT (1st ORDER)

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

void plotpixel(int x, int y)
{
//  delay(10);
  putpixel((x+320),(240-y),15);
  putpixel((-x+320),(240-y),15);
  putpixel((x+320),(240+y),15);
  putpixel((-x+320),(240+y),15);

}

void ell(float a, float b)
{
  float x=0,y=b;

  float d = a*a*(0.25-b) + b*b;

  setcolor(RED);
  line(320,0,320,480);
  setcolor(BLUE);
  line(0,240,640,240);
  setcolor(WHITE);

  plotpixel(x,y);

  while((b*b*(x+1))<=(a*a*(y-0.5)))
  {
   if(d<0)
   {
    d+= 1.0*b*b*(3+2*x);
    x++;
   }
   else
   {
    d+= b*b*(3+2*x)*1.0+2*1.0*a*a*(1-y);
    x++;
    y--;
   }
   plotpixel(x,y);
  }

  d = b*b*(1.0*x+0.5)*(1.0*x+0.5)+1.0*a*a*(y-1)*(y-1)-1.0*a*a*b*b;

  while(y>=0)
  {
   if(d<0)
   {
```

```
      d+= 2.0*b*b*(1+x)+1.0*a*a*(3-2*y);
      x++;
      y--;
    }
    else
    {
      d+= a*a*(3-2*y)*1.0;
      y--;
    }
    plotpixel(x,y);
  }

}

void main()
{
  clrscr();

  float a,b;
  cout<<"Enter a & b : ";
  cin>>a>>b;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

  ell(a,b);
  getch();
  closegraph();
}
```

OUTPUT

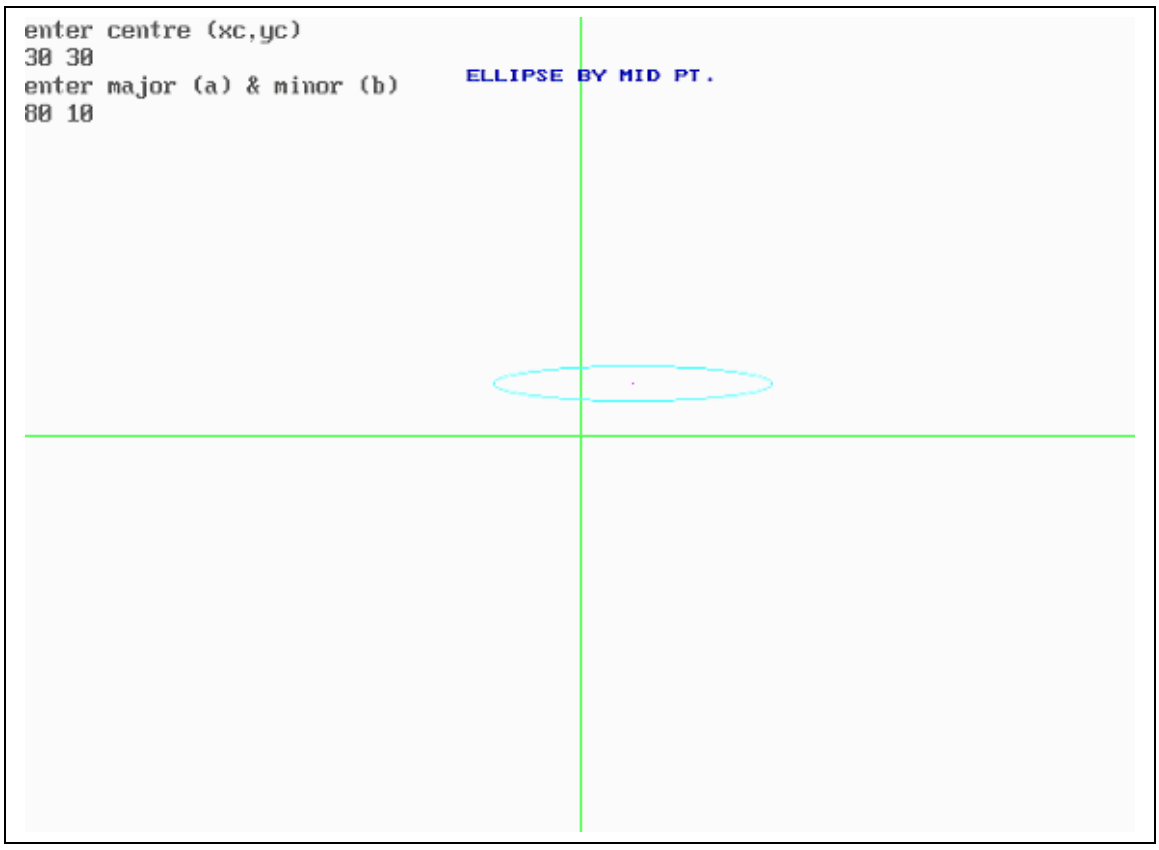

enter centre (xc,yc)
30 30
enter major (a) & minor (b)
80 10
ELLIPSE BY MID PT.

ELLIPSE - MID POINT (2nd ORDER)

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

void plotpixel(int x, int y)
{
//  delay(10);
  putpixel((x+320),(240-y),15);
  putpixel((-x+320),(240-y),15);
  putpixel((x+320),(240+y),15);
  putpixel((-x+320),(240+y),15);

}

void ell(float a, float b)
{
  float x=0,y=b;

  float d = a*a*(0.25-b) + b*b;
  float de = 3*b*b;
  float dse = 3*b*b + a*a*(2-2*b);

  setcolor(RED);
  line(320,0,320,480);
  setcolor(BLUE);
  line(0,240,640,240);
  setcolor(WHITE);

  plotpixel(x,y);

  while((b*b*(x+1))<=(a*a*(y-0.5)))
  {
   if(d<0)
   {
     d+= de;
     de+=2*b*b;
     dse+=2*b*b;
     x++;
   }
   else
   {
     d+= dse;
     de+=2*b*b;
     dse+=2*(a*a+b*b);
     x++;
     y--;
   }
   plotpixel(x,y);
  }

  d = b*b*(1.0*x+0.5)*(1.0*x+0.5)+1.0*a*a*(y-1)*(y-1)-1.0*a*a*b*b;
```

```
  float ds = a*a*(3-2*y);
  dse = 2*b*b*(1+x) + a*a*(3-2*y);

  while(y>=0)
  {
    if(d<0)
    {
      d+= dse;
      ds+=2*a*a;
      dse+=2*(a*a+b*b);
      x++;
      y--;
    }
    else
    {
      d+= ds;
      ds+=2*a*a;
      dse+=2*a*a;
      y--;
    }
    plotpixel(x,y);
  }

}

void main()
{
  clrscr();

  float a,b;
  cout<<"Enter a & b : ";
  cin>>a>>b;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

  ell(a,b);
  getch();
  closegraph();
}
```

OUTPUT


enter centre (xc,yc)
30 30
enter major (a) & minor (b)
80 10
ELLIPSE BY MID PT.

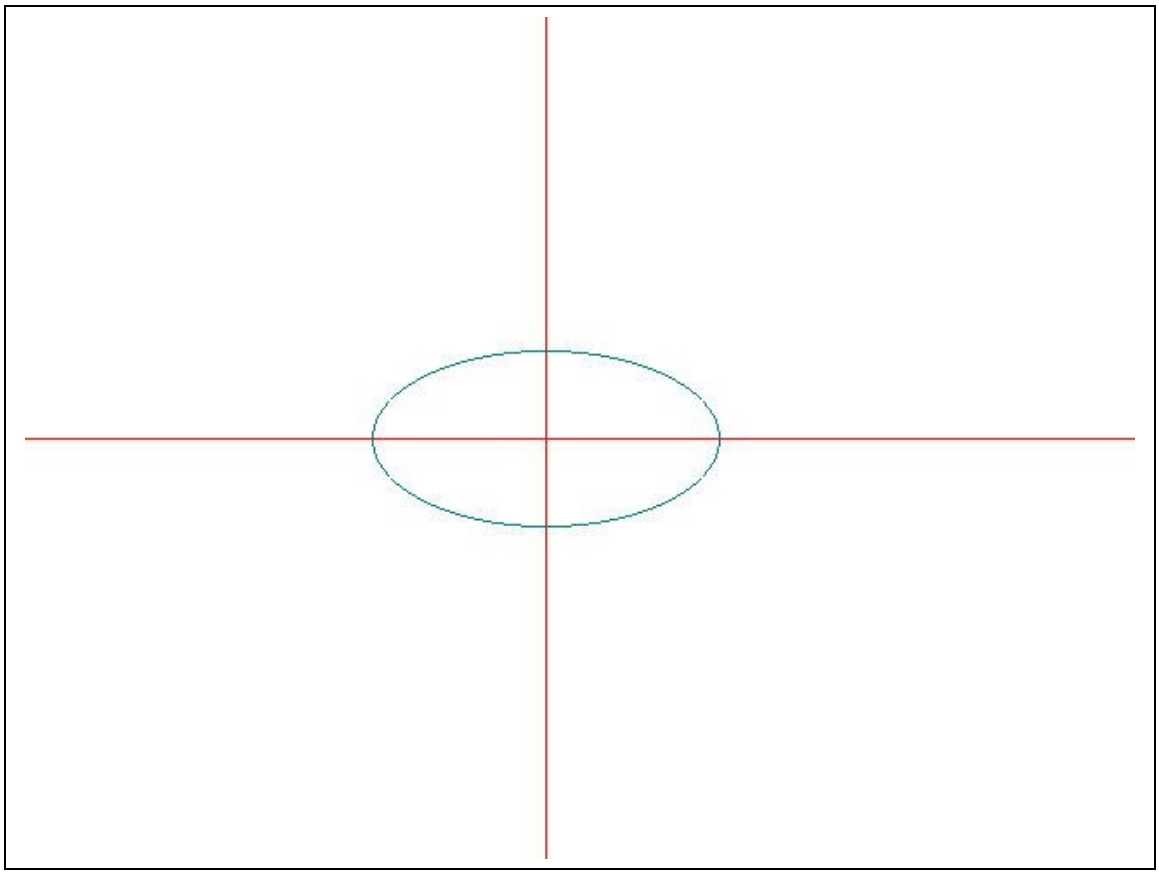
ELLIPSE – BRESENHAM'S APPROACH

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

void plotpixel(int x, int y)
{
 putpixel((x+300),(240-y),15);
 putpixel((-x+300),(240-y),15);
 putpixel((x+300),(240+y),15);
 putpixel((-x+300),(240+y),15);
}

void ellip(int cx, int cy, double a,double b)
{
 setcolor(BLUE);
 line(300,0,300,479);
 setcolor(RED);
 line(0,240,639,240);
 double x=0,y=b;  /* initial coorodinates */

 double d1 = 2*(b*b) + (a*a*(1 - 2*b));
 plotpixel(x,y);

 while((a*a*y) >= (b*b*x))
 {
  if( d1<= (a*a*0.5) )
  {
   d1+= (4*b*b*x + 6*b*b);
   x++;
  }
  else
  {
   d1+= ((4*b*b*x + 6*b*b) - 4*a*a*(y-1));
   x++;
   y--;
  }
  plotpixel(x,y);
 }

 d1 = (b*b*(x+1)*(x+1)) + b*b*x*x - 2*a*a*b*b + 2*(a*a*(y-1)*(y-1));

 while(y>=0)
 {
  if(d1 >= (b*b*0.5))
  {
   d1+= 6*a*a - 4*a*a*y;
   y--;
  }
  else
  {
   d1+= (b*b*4*(1+x)) + 2*a*a - (4*a*a*(y-1));
```

```
      x++;
      y--;
    }
    plotpixel(x,y);
  }

}

void main()
{
  clrscr();
  double a,b;
  cout<<"Enter a and b : ";
  cin>>a>>b;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  ellip(0,0,a,b);
  getch();
  closegraph();
}
```

OUTPUT

Enter a and b: 100 50



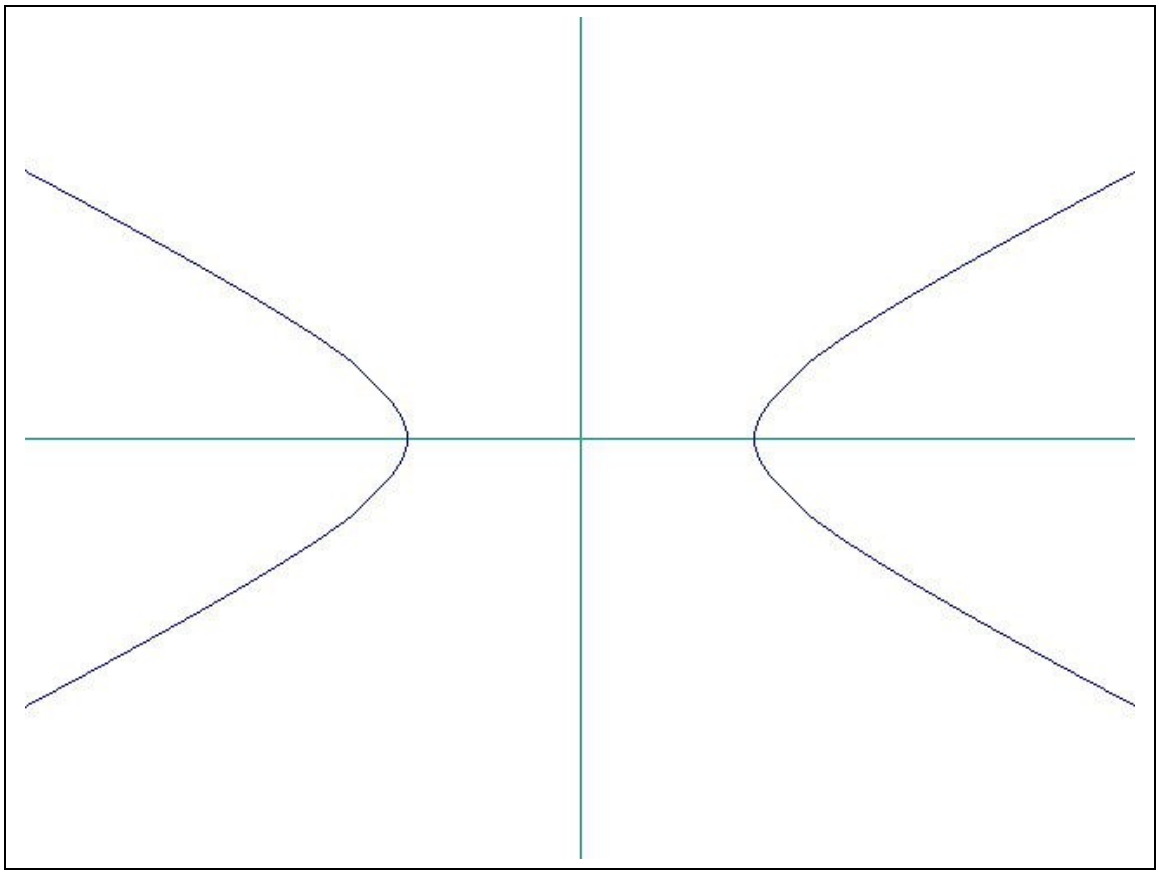
HYPERBOLA - MID POINT (1st ORDER)

```
#include<graphics.h>
#include<conio.h>
#include<stdio.h>
#include<dos.h>
#include<stdlib.h>

void main()
{
 int gd=DETECT;
 int gm;
 initgraph(&gd,&gm,"c:\\tc\\bgi");
 float x,y;
 float xc,yc;
 float d,fx,fy,b,a;
 d=b*b*(a+0.5)*(a+0.5)-a*a-a*a*b*b;
 printf("enter centre (xc,yc)\n");
 scanf("%f %f",&xc,&yc);
 printf("enter a & b");
 scanf("%f %f",&a,&b);
 x=a;
 y=0;
 fx=2*b*b*a;
 fy=0;
 setcolor(MAGENTA);
 line(0,240,640,240);
 line(320,0,320,480);
 while(abs(fy)<=fx)
{
  if(d>=0)
 {
   d=d-a*a*(2*y+3);
 }
 else
 {
   d=d-a*a*(2*y+3)+b*b*(2*x+2);
   x++;
   fx=fx+2*b*b;
 }
 y++;
 fy=fy+2*a*a;
 putpixel(x+320+xc,240-y-yc,GREEN);
 putpixel(x+320+xc,240+y-yc,GREEN);
 putpixel(-x+320+xc,240-y-yc,GREEN);
 delay(20);
 putpixel(-x+320+xc,240+y-yc,GREEN);
 delay(20);
}
x=p/2;
y=p;
d=-p;
while(y<3*p)
{
 x++;
```

```
 if(d>=0)
 {
   d=d-2*p;
 }
 else
 {
  d=d+2*y+2-2*p;
  y++;
 }
 putpixel(x+320+xc,240-y-yc,RED);
 delay(20);
 putpixel(x+320+xc,240+y-yc,RED);
 delay(20);
}
getch();
}
```

OUTPUT

Enter a and b : 100 99



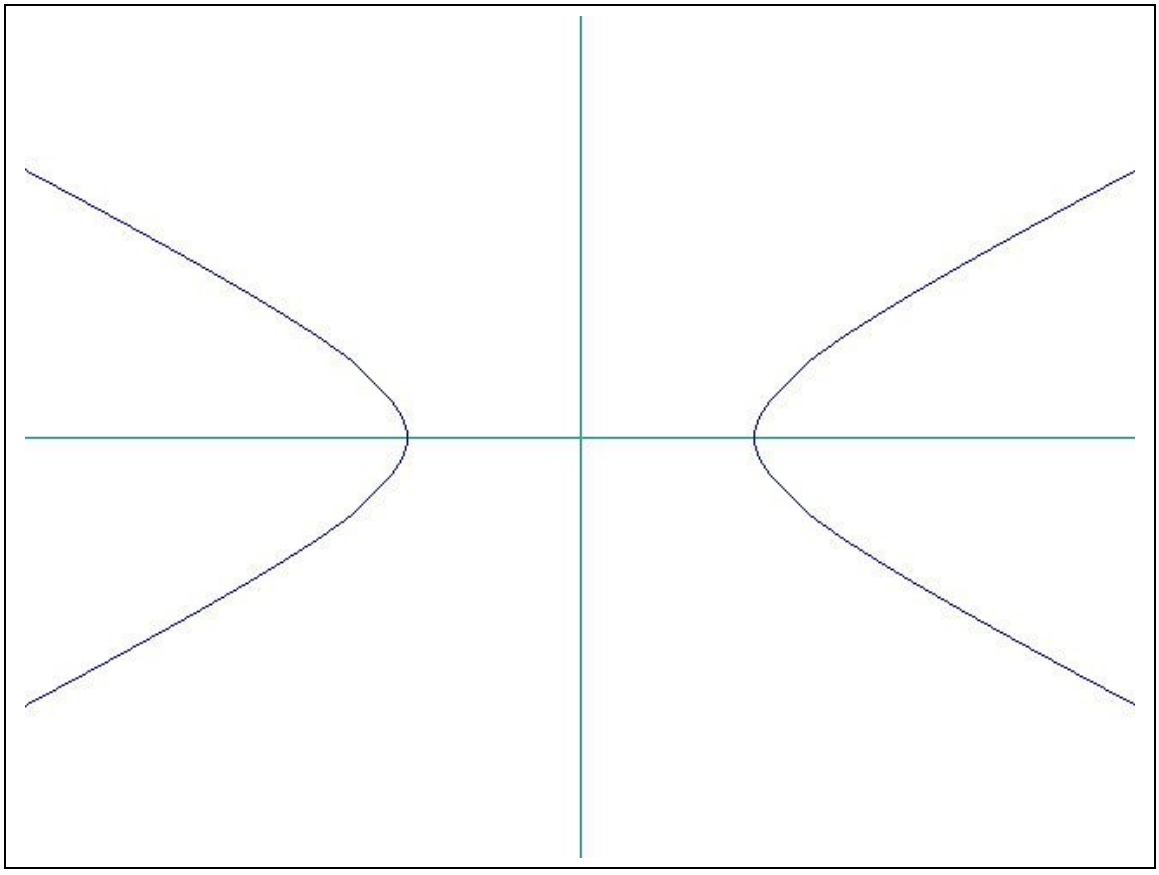
HYPERBOLA – BRESENHAM'S APPROACH

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

void plotpixel(int x, int y)
{
 putpixel((x+300),(240-y),15);
 putpixel((-x+300),(240-y),15);
 putpixel((x+300),(240+y),15);
 putpixel((-x+300),(240+y),15);
}

void hyper(int cx, int cy, double a,double b)
{
 setcolor(BLUE);
 line(300,0,300,479);
 setcolor(RED);
 line(0,240,639,240);
 double x=a,y=0;  /* initial coorodinates */

 double d1 = (2*a*a) - (b*b) - (2*a*b*b);
 plotpixel(x,y);

 while((a*a*y) <= (b*b*x))
 {
  if(d1<= (-1*b*b*0.5))
  {
   d1+= 2*a*a*(2*y+3);
   plotpixel(x,y);
   y++;
  }
  else
  {
   d1+= 2*a*a*(2*y+3) - 4*b*b*(x+1);
   plotpixel(x,y);
   x++;
   y++;
  }
  //plotpixel(x,y);
 }

 d1 = a*a*(y+1)*(y+1) + a*a*y*y + 2*a*a*b*b - 2*a*a*b*b*(x+1)*(x+1);

 while(y<220)
 {
  if(d1<= (a*a*0.5))
  {
   d1+= a*a*4*(y+1) - 2-a*a*b*b*(2*x+3)*(2*x+3);
   y++;
   x++;
  }
```

```
   else
   {
     d1+= -2.0*b*b*a*a*(2*x+3);
     x++;
   }
   plotpixel(x,y);
  }

}

void main()
{
  clrscr();
  double a,b;
  cout<<"Enter a and b : ";
  cin>>a>>b;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  hyper(0,0,a,b);
  getch();
  closegraph();
}
```

OUTPUT

Enter a and b: 100 99



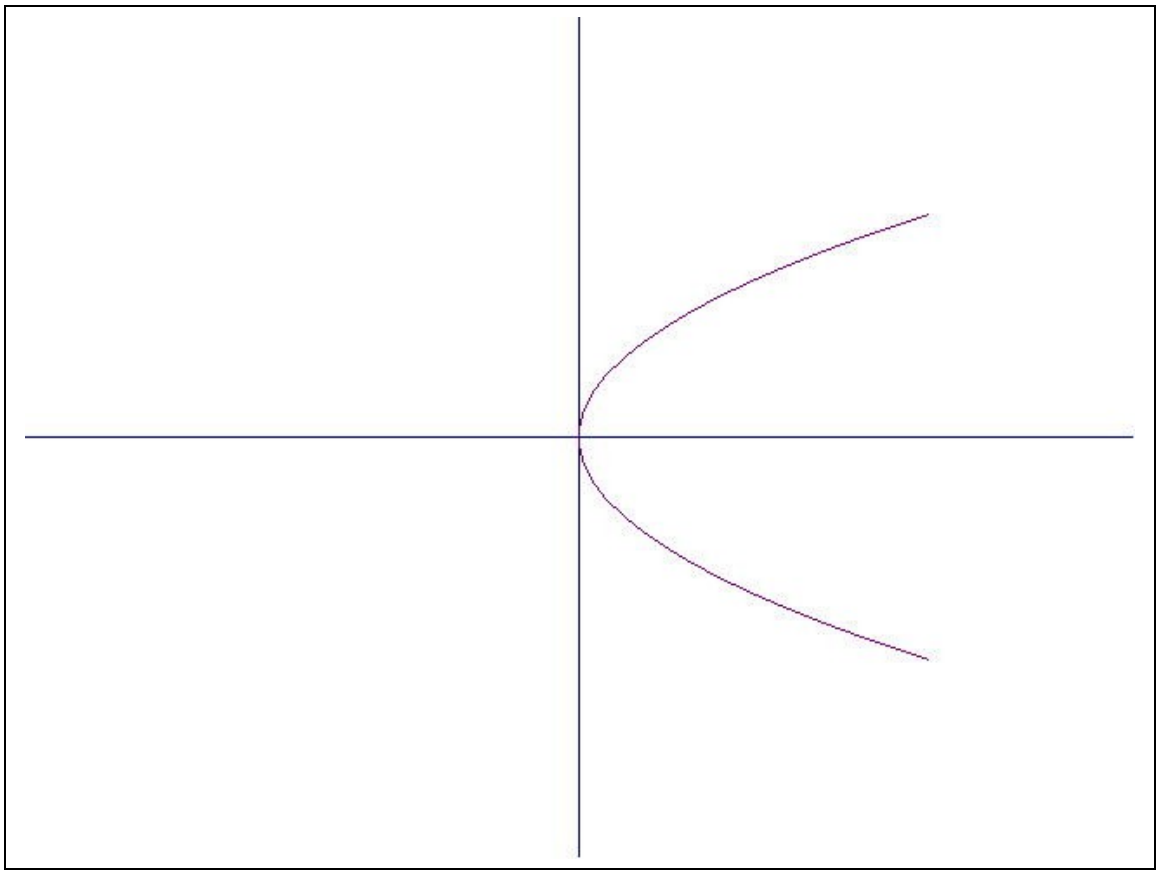
PARABOLA – MID POINT (1st ORDER)

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

void put_pixel(int x, int y)
{
 putpixel((x+300),(240-y),15);
 putpixel((x+300),(240+y),15);
}

void para(int cx, int cy, double a)
{
 setcolor(BLUE);
 line(300,0,300,479);
 setcolor(RED);
 line(0,240,639,240);
 double x=0,y=0;  /* initial coorodinates */

 double d1;
 d1 = (2*a) - 1;
 put_pixel(x,y);

 while(y<= (2*a*1.0))
 {
  if(d1<0)
  {
   d1+= 4*a-3-2*y;
   x++;
   y++;
  }
  else
  {
   d1-= 3 + 2*y;
   y++;
  }
  put_pixel(x,y);
 }

 d1 = (4.0*a*(x+1) - (y+0.5)*(y+0.5) );

 while( y < 220 )
 {
  if(d1<0)
  {
   d1+= 4*a;
   x++;
  }
  else
  {
   d1+= 4.0*a - 2 - 2.0*y ;
   x++;
   y++;
```

```
    }
    put_pixel(x,y);
  }

}

void main()
{
  clrscr();
  double a;
  cout<<"Enter a : ";
  cin>>a;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  para(0,0,a);
  getch();
  closegraph();
}
```

OUTPUT

Enter a : 50



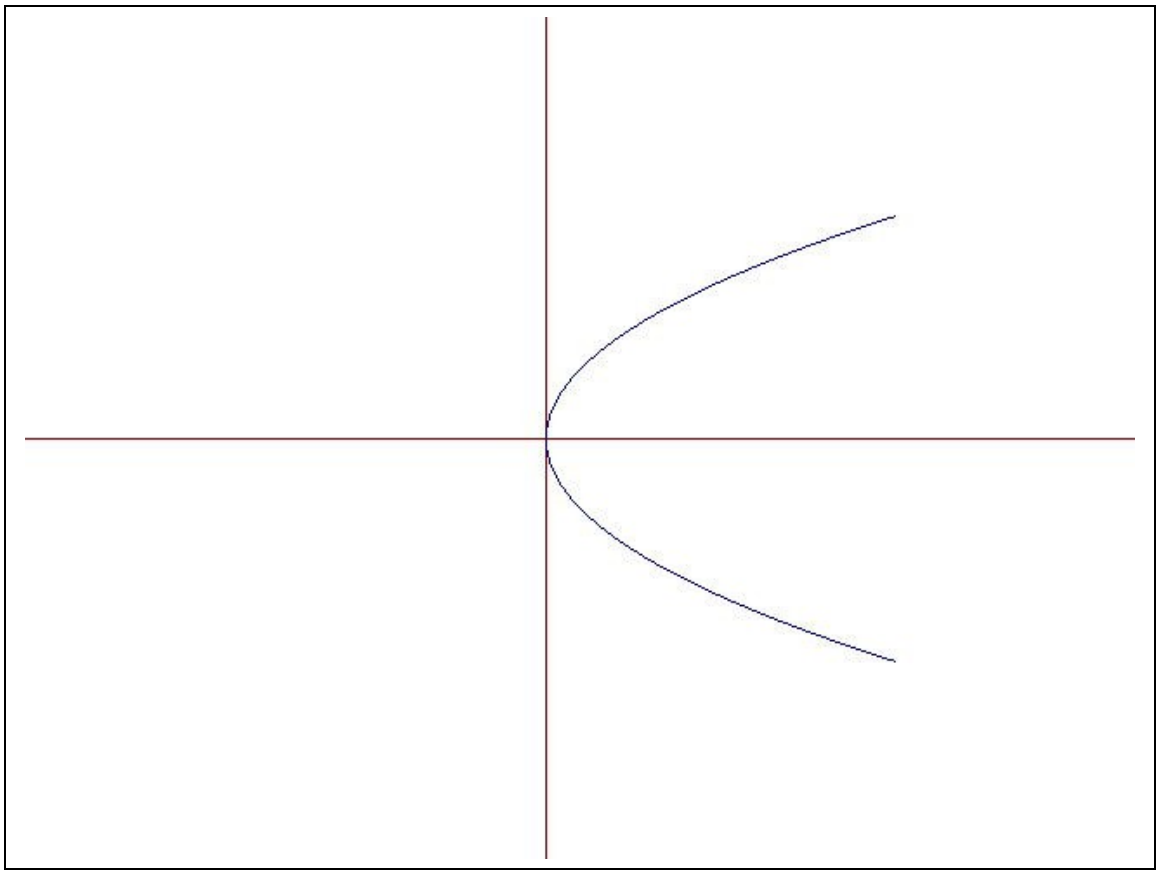
PARABOLA – MID POINT (2nd ORDER)

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

void put_pixel(int x, int y)
{
 putpixel((x+300),(240-y),15);
 putpixel((x+300),(240+y),15);
}

void para(int cx, int cy, double a)
{
 setcolor(BLUE);
 line(300,0,300,479);
 setcolor(RED);
 line(0,240,639,240);
 double x=0,y=0;  /* initial coorodinates */

 double d1;
 d1 = (2*a) - 1;
 double dne = 4*a-3;
 double dn = -3;
 put_pixel(x,y);

 while(y<= (2*a*1.0))
 {
  if(d1<0)
  {
   d1+= dne;
   dne-=2;
   dn-=2;
   x++;
   y++;
  }
  else
  {
   d1+=dn;
   dne-=2;
   dn-=2;
   y++;
  }
  put_pixel(x,y);
 }

 d1 = (4.0*a*(x+1) - (y+0.5)*(y+0.5) );
 double de = 4*a;
 dne = 4*a - 2*(1+y);
 while( y < 220 )
 {
  if(d1<0)
  {
```

```
      d1+= de;
      x++;
    }
    else
    {
      d1+= dne;
      dne+=-2;
      x++;
      y++;
    }
    put_pixel(x,y);
  }

}

void main()
{
  clrscr();
  double a;
  cout<<"Enter a : ";
  cin>>a;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  para(0,0,a);
  getch();
  closegraph();
}
```

OUTPUT

Enter a : 50

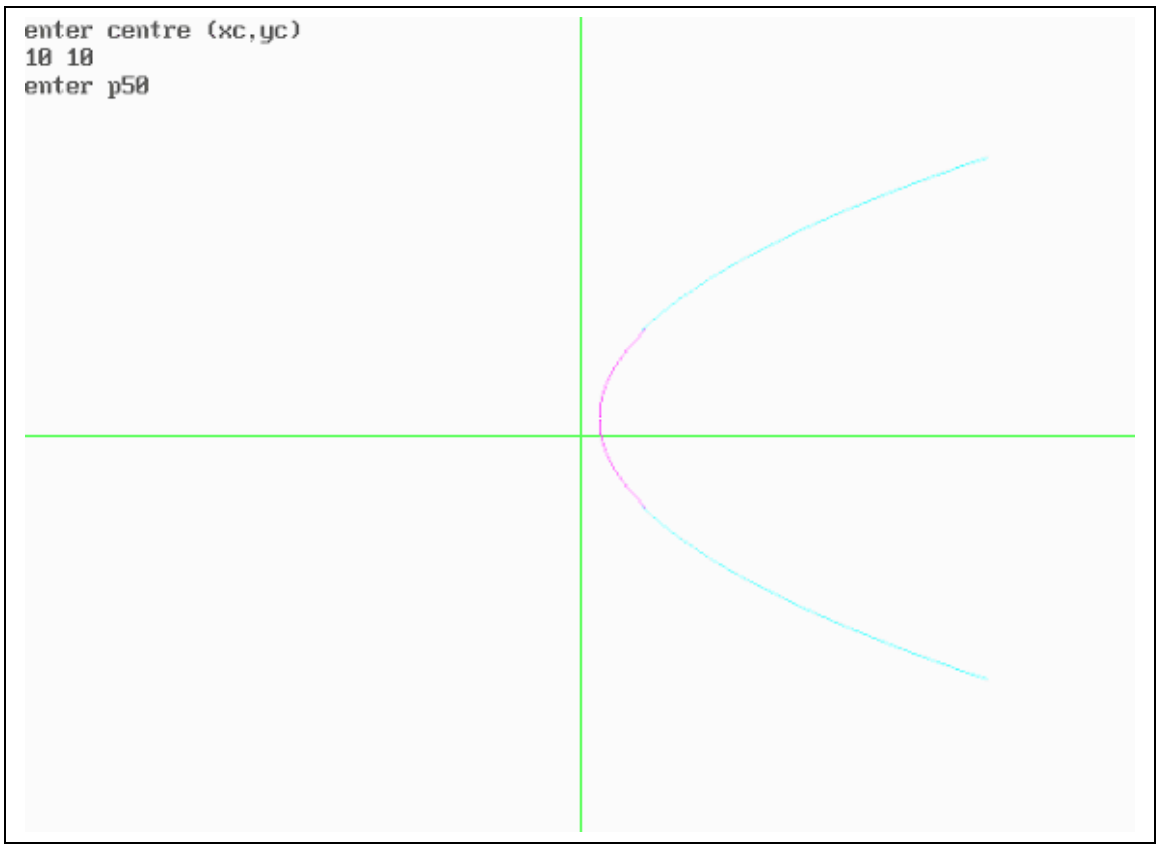


PARABOLA – BRESENHAM'S APPROACH

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

void plotpixel(int x, int y)
{
 putpixel((x+300),(240-y),15);
//  putpixel((-x+300),(240-y),15);
 putpixel((x+300),(240+y),15);
//  putpixel((-x+300),(240+y),15);
}

void para(int cx, int cy, double a)
{
 setcolor(BLUE);
 line(300,0,300,479);
 setcolor(RED);
 line(0,240,639,240);
 double x=0,y=0;  /* initial coorodinates */

 double d1 = 1 - (2*a);
 plotpixel(x,y);

 while( y<= (2*a) )
 {
  if(d1>0)
  {
   d1+= 3 + 2*y - 4*a;
   plotpixel(x,y);
   x++;
   y++;
  }
  else
  {
   d1+= 3 + 2*y;
   plotpixel(x,y);
   y++;
  }
  //plotpixel(x,y);
 }

 d1 = ((y+0.5)*(y+0.5) - 4*a*(x+1));

 while( y < 220 )
 {
  if(d1>0)
  {
   d1+= (-4*a);
   x++;
  }
  else
```

```
    {
      d1+= 2 + 2*y - 4*a;
      x++;
      y++;
    }
    plotpixel(x,y);
  }

}

void main()
{
  clrscr();
  double a;
  cout<<"Enter a : ";
  cin>>a;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  para(0,0,a);
  getch();
  closegraph();
}
```

OUTPUT



LINE CLIPPING - COHEN SUTHERLAND

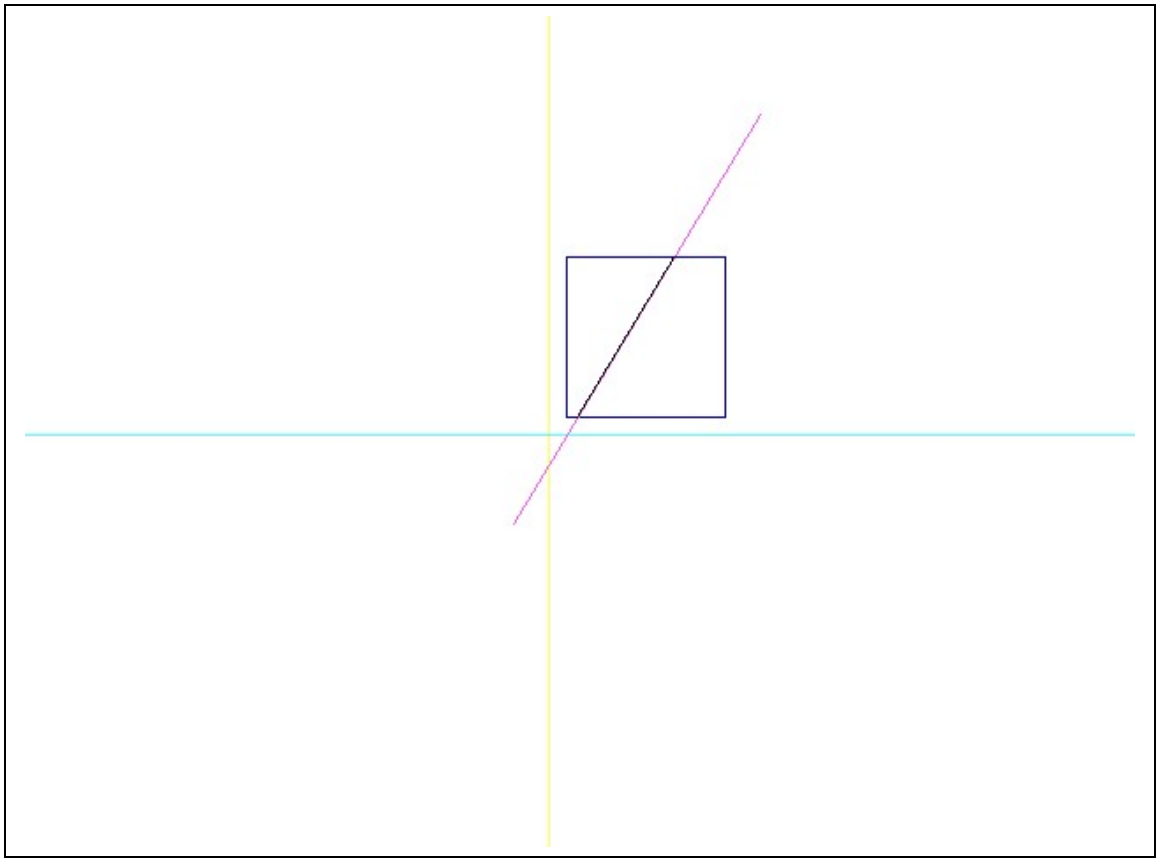
```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>

#define LEFT 0x01
#define RIGHT 0x4
#define BOTTOM 0x2
#define TOP 0x8

char getcode(float x, float y, float xwmin, float ywmin, float xwmax, float ywmax)
{
 unsigned char code = 0x00;
 if(x<xwmin)
   code = code|LEFT;
 if(x>xwmax)
   code = code|RIGHT;
 if(y>ywmin)
   code = code|BOTTOM;
 if(y<ywmax)
   code = code|TOP;
 return code;
}

void lin(float x1, float y1, float x2, float y2, float xwmin, float ywmin, float xwmax, int ywmax)
{
 int done = 0, accept = 0;
 unsigned char code1, code2;
 int gdriver = DETECT, gmode;
 initgraph(&gdriver,&gmode,"c:\\tc\\bgi");

 setcolor(BLUE);
 line(300,0,300,479);
 setcolor(RED);
 line(0,240,639,240);

 setcolor(YELLOW);
 rectangle(xwmin, ywmin, xwmax, ywmax);
 setcolor(GREEN);
 line(x1,y1,x2,y2);
 getch();

 setcolor(WHITE);
 float m;
 while(done==0)
 {
  code1 = getcode(x1,y1,xwmin,ywmin,xwmax,ywmax);
  code2 = getcode(x2,y2,xwmin,ywmin,xwmax,ywmax);

  /* case I - accept line */
  if(((code1&code2)==0) && ((code1|code2)==0))
  {
    accept = 1;
```

```
    done = 1;
   }
   else if((code1&code2)!=0)
   {
    done = 1;
    outtextxy(10,300,"\n Sorry! Line rejected");
   }
   else
   {
    if((x1>= xwmin && x1<= xwmax) && (y1>= ywmax && y1<=ywmin))
    {
          float temp = x1;
          x1 = x2;
          x2=temp;
          temp = y1;
          y1=y2;
          y2=temp;
          char t;
          t=code1;
          code1=code2;
          code2=t;
    }

    if(x1!=x2)
          m = (y2-y1)/(x2-x1);

    if( code1 & LEFT != 0)
    {
          y1+= (xwmin-x1)*m;
          x1 = xwmin;
    }
    else if(code1 & RIGHT)
    {
          y1+= (xwmax-x1)*m;
          x1 = xwmax;
    }
    else if(code1 & BOTTOM)
    {
          if(x2!=x1)
            x1+= (ywmin - y1)/m;
          y1 = ywmin;
    }
    else
    {
          if(x2!=x1)
            x1+= (ywmax-y1)/m;
          y1 = ywmax;
    }
   }
  }
  if(accept == 1)
    line(x1,y1,x2,y2);
}

void main()
{
```

```
 int gdriver = DETECT, gmode;
 float xwmin, xwmax, ywmin, ywmax;
 cout<<" Enter the x limits for the clipping window : ";
 cin>>xwmin>>xwmax;
 cout<<"\n Enter the y limits fro the clipping window :";
 cin>>ywmin>>ywmax;

 cout<<"\n Enter end point 1 : ";
 float x1,y1,x2,y2;
 cin>>x1>>y1;
 cout<<"\n Enter end point 2 : ";
 cin>>x2>>y2;

 lin(x1+300,240-y1,x2+300,240-y2,xwmin+300,240-ywmin,xwmax+300,240-ywmax);
 //  i have interchanged ywmin and ywmax here beacuse we are doing 240-y
 getch();
 closegraph();
}
```

OUTPUT

```
Enter the x limits for the clipping window : 10 100

Enter the y limits fro the clipping window :10 100

Enter end point 1 : -20 -50

Enter end point 2 : 120 180
```



LINE CLIPPING - LIANG BARSKEY

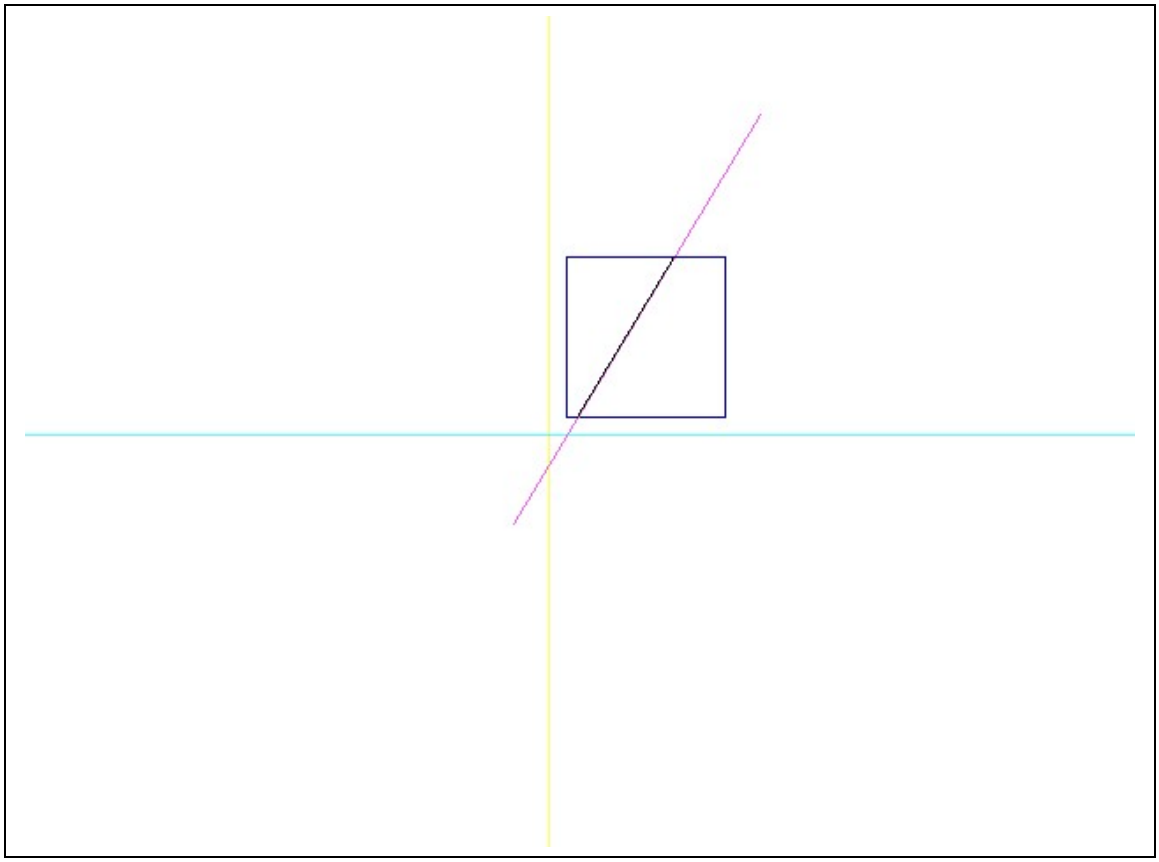
```
#include<graphics.h>
#include<conio.h>
#include<iostream.h>
#include<process.h>

float max(float a,float b,float c,float d)
{
 float e,f;
 if(a>b)
 {
  e=a;
 }
 else
 {
  e=b;
 }
 if(c>d)
 {
  f=c;
 }
 else
 {
  f=d;
 }
 if(e>f)
 {
  return e;
 }
 else
 {
  return f;
 }
}

float min(float g,float h,float m,float j)
{
 float k,l;
 if(g<h)
 {
  k=g;
 }
 else
 {
  k=h;
 }
 if(m<j)
 {
  l=m;
 }
 else
 {
  l=j;
```

```
 }
 if(k<l)
 {
  return k;
 }
 else
 {
  return l;
 }
}

void main()
{
 int gd=DETECT;
 int gm;
 initgraph(&gd,&gm,"c:\\tc\\bgi");
 float x1,x2,y1,y2,u[4]={0},p[10],i;
 float q[10],dx,dy,xmin,xmax,ymin,ymax;
 float xm,ym,xn,yn;
 cout<<"enter x1,y1,x2,y2,xmin,xmax,ymin,ymax";
 cin>>x1>>y1>>x2>>y2>>xmin>>xmax>>ymin>>ymax;
 setcolor(BLUE);
 line(320+xmin,240-ymin,320+xmax,240-ymin);
 line(320+xmin,240-ymin,320+xmin,240-ymax);
 line(320+xmax,240-ymax,320+xmax,240-ymin);
 line(320+xmin,240-ymax,320+xmax,240-ymax);
 setcolor(GREEN);
 line(320+x1,240-y1,320+x2,240-y2);
 getch();
 dx=x2-x1;
 dy=y2-y1;
 p[1]=-dx;p[2]=dx;p[3]=-dy;p[4]=dy;
 q[1]=x1-xmin;
 q[2]=xmax-x1;
 q[3]=y1-ymin;
 q[4]=ymax-y1;
 float u1=0.0;
 float u2=1.0;
 for(i=1;i<=4;i++)
 {
  if(p[i]<0 )
  u[i]=q[i]/p[i];
 }
 u1=max(u[1],u[2],u[3],u[4]);
 if(u1<0)
 {
  u1=0;
 }
 u[1]=2;u[2]=2;u[3]=2;u[4]=2;
 for(i=1;i<=4;i++)
 {
  if(p[i]>0)
  u[i]=q[i]/p[i];
 }
 u2=min(u[1], u[2],u[3],u[4]);
 if(u2>1)
```

```
 {
  u2=1;
 }
 if(p[1]==0)
 {
  if(q[1]<0 ||q[2]<0)
  {
   exit(0);
  }
 }
 if(u1>u2)
 {
  exit(0);
 }
 xm=x1+u1*dx;
 ym=y1+u1*dy;
 xn=x1+u2*dx;
 yn=y1+u2*dy;
 setcolor(WHITE);
 line(320+xm,240-ym,320+xn,240-yn);
 getch();
}
```

OUTPUT

```
Enter the x limits for the clipping window : 10 100

Enter the y limits fro the clipping window :10 100

Enter end point 1 : -20 -50

Enter end point 2 : 120 180
```



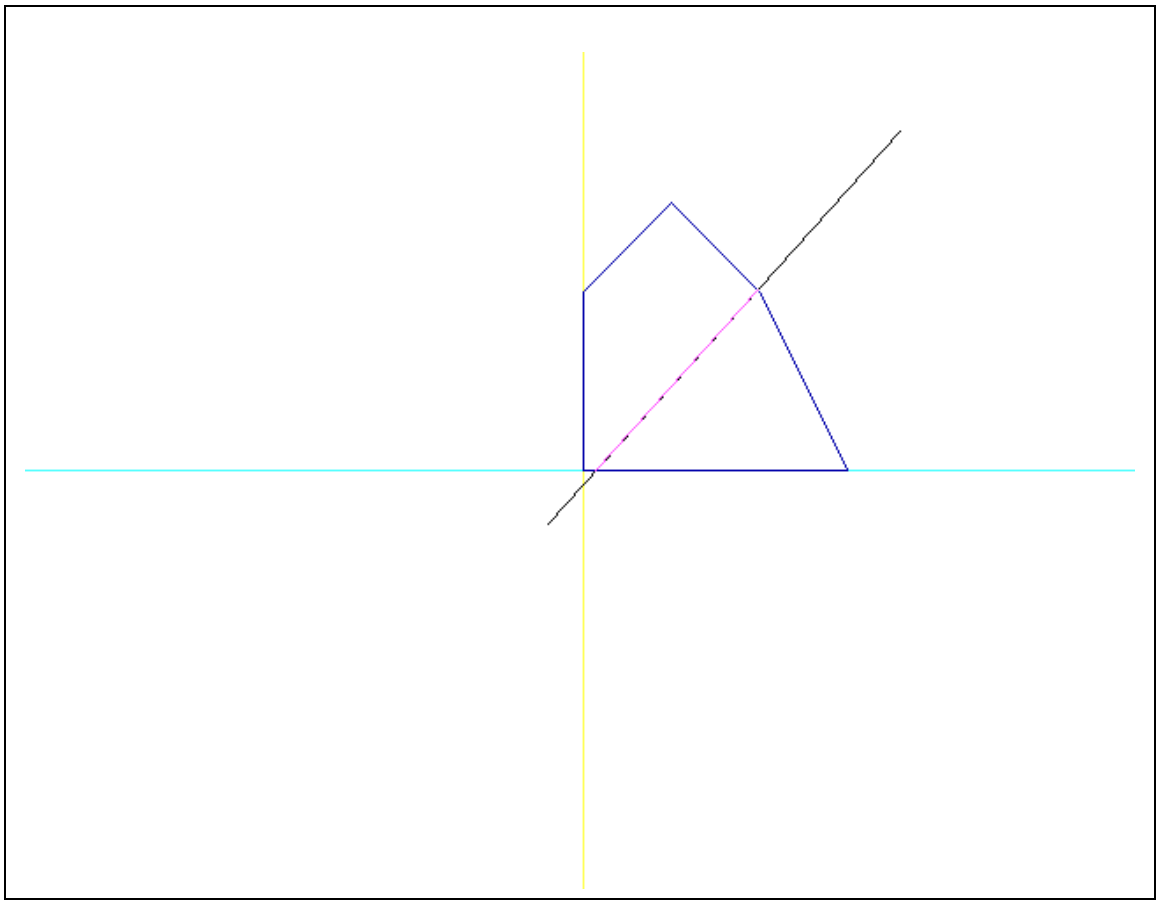
LINE CLIPPING - CYRUS BECK

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

struct point
{
 float x,y;
};

void clip(point pol[10], point p1, point p2, int n)
{
 cleardevice();

 setcolor(BLUE);
 line(320,0,320,480);
 setcolor(RED);
 line(0,240,640,240);

 setcolor(YELLOW);
 for(int i=0;i<n;i++)
 {
  line(pol[i].x,pol[i].y,pol[i+1].x,pol[i+1].y);
 }
 setcolor(WHITE);
 line(p1.x,p1.y,p2.x,p2.y);
 getch();
 float t_enter=0,t_leave=1;
 for(i=0;i<n;i++)
 {
  point n,pei;
  pei=pol[i];
  n.x=(pol[i+1].y-pol[i].y);
  n.y=(pol[i+1].x-pol[i].x);
  float num,den;
  num = n.x*(pei.x-p1.x) - n.y*(pei.y-p1.y);
  den = n.x*(p2.x-p1.x) + n.y*(p1.y-p2.y);
  float t;
  if(den!=0)
    t= num*1.0/den;

  if(t>=0 && t<=1)
  {
   if(den<0)
   {
          if(t>t_enter)
            t_enter = t;
   }
   else if(den>0)
   {
          if(t<t_leave)
            t_leave = t;
   }
```

```
    }
  }

  point pi,pl;
  pi.x=p1.x+(p2.x-p1.x)*t_enter;
  pi.y=p1.y+(p2.y-p1.y)*t_enter;
  pl.x=p1.x+(p2.x-p1.x)*t_leave;
  pl.y=p1.y+(p2.y-p1.y)*t_leave;
  setcolor(GREEN);
  line(pi.x,pi.y,pl.x,pl.y);
}

void main()
{
  int gd = DETECT, gm;
  initgraph(&gd,&gm,"c:\\tc\\bgi");

  cout<<"Enter the no. of vertices of clipping window : ";
  int n;
  cin>>n;

  point pol[10];
  cout<<"Enter the vertices in clockwise order \n";
  for(int i=0;i<n;i++)
  {
    cout<<" Enter vertex : ";
    cin>>pol[i].x>>pol[i].y;
    pol[i].x+=320;
    pol[i].y=240-pol[i].y;
  }
  pol[i].x=pol[0].x;
  pol[i].y=pol[0].y;

  cout<<"Enter the end points of the line : ";
  point p1,p2;
  cin>>p1.x>>p1.y>>p2.x>>p2.y;

  int t;
  if(p1.x>p2.x)
  {
    t=p1.x;
    p1.x=p2.x;
    p2.x=t;
    t=p1.y;
    p1.y=p2.y;
    p2.y=t;
  }

  p1.x+=320;p2.x+=320;
  p1.y=240-p1.y;
  p2.y=240-p2.y;

  clip(pol,p1,p2,n);
  getch();
  closegraph();
}
```

OUTPUT

```
Enter the no. of vertices of clipping window : 5
Enter the vertices in clockwise order
 Enter vertex : 0 0
 Enter vertex : 0 100
 Enter vertex : 50 150
 Enter vertex : 100 100
 Enter vertex : 150 0
Enter the end points of the line : -20 -30 180 190
```



MID POINT SUBDIVISION METHOD

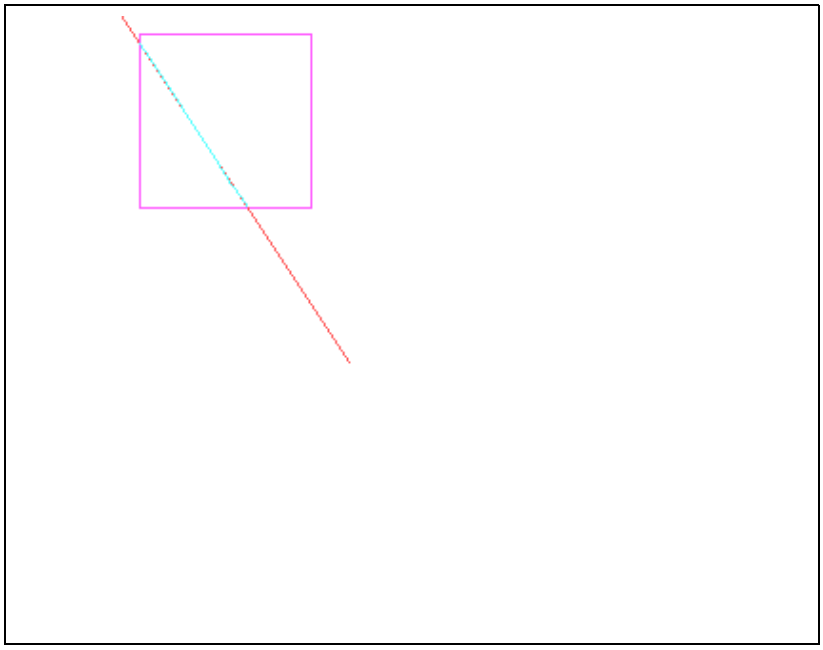
```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>

#define LEFT 0x01
#define RIGHT 0x4
#define BOTTOM 0x2
#define TOP 0x8


struct point
{
 float x,y;
};

char getcode(point p, point wmin, point wmax)
{
 unsigned char code = 0x00;
 if(p.x<wmin.x)
   code = code|LEFT;
 if(p.x>wmax.x)
   code = code|RIGHT;
 if(p.y>wmax.y)
   code = code|BOTTOM;
 if(p.y<wmin.y)
   code = code|TOP;
 return code;
}

int isin(point p, point wmin, point wmax)
{
 char stcode =0x00;
 char cod = getcode(p,wmin,wmax);
 if(((cod&stcode)==0) && ((cod|stcode)==0))
   return 1;
 else
   return 0;
}

point mid( point p1, point p2)
{
 point m;
 m.x = (p1.x+p2.x)/2;   // calculate mid point of the line segment
 m.y = (p1.y+p2.y)/2;

 return m;
}

float dist( point t1, point t2, point wmin, point wmax)
{
 float dsx = t1.x - t2.x;
```

```
 float dsy = t1.y - t2.y;
 /*float ds;
 ds  = sqrt((dsx*dsx) + (dsy*dsy));
 return ds;*/
 float ds = dsx>dsy?dsx:dsy;
 return ds;
}

void midpt(point wmin, point wmax, point p1, point p2)
{
 setcolor(RED);
 point t;
 if (p1.x > p2.x)
 {
  t = p1;
  p1 = p2;
  p2 = t;
 }

 char code1 = getcode(p1,wmin,wmax);
 char code2 = getcode(p2,wmin,wmax);

 if((code1 & code2)!=0)
 {
  return;
 }
 else if(((code1 & code2)==0) && ((code1 | code2)==0))
 {
   line(p1.x, p1.y, p2.x, p2.y);
 }
 else
 {
   t = mid(p1,p2);
   if((dist(t,p1,wmin,wmax) >= 1) || (dist(t,p2,wmin,wmax) >= 1))
   {
    char codm = getcode(t, wmin, wmax);

    if(isin(t,wmin,wmax))
    {
          if(isin(p1,wmin,wmax))
          {
           line(p1.x,p1.y,t.x,t.y);
           p1=t;
           midpt(wmin,wmax,p1,p2);
          }
          else if(isin(p2,wmin,wmax))
          {
           line(p2.x,p2.y,t.x,t.y);
           p2=t;
           midpt(wmin,wmax,p1,p2);
          }

          else
          {
           midpt(wmin,wmax,p1,t);
           midpt(wmin,wmax,t,p2);
```

```
            }
        }
        else
        {
            if ((code1&codm)!=0)
            {
              p1=t;
            }
            else if((code2&codm)!=0)
            {
              p2=t;
            }
            midpt(wmin,wmax,p1,p2);
        }
      }
    }
}

void main()
{
  clrscr();
  point wmin,wmax,p1,p2;
  cout<<" Enter the clipping window limits : Xwmin  ";
  cin>>wmin.x;
  cout<<" Enter the clipping window limits : Ywmin  ";
  cin>>wmin.y;
  cout<<" Enter the clipping window limits : Xwmax  ";
  cin>>wmax.x;
  cout<<" Enter the clipping window limits : Ywmax  ";
  cin>>wmax.y;

  cout<<"\n Enter the end points of the line : P1->x  ";
  cin>>p1.x;
  cout<<"\n Enter the end points of the line : P1->y  ";
  cin>>p1.y;
  cout<<"\n Enter the end points of the line : P2->x  ";
  cin>>p2.x;
  cout<<"\n Enter the end points of the line : P2->y  ";
  cin>>p2.y;

  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  setcolor(GREEN);
  rectangle(wmin.x,wmin.y,wmax.x,wmax.y);
  setcolor(CYAN);
  line(p1.x,p1.y,p2.x,p2.y);
  getch();
  midpt(wmin,wmax,p1,p2);
  getch();
  closegraph();
}
```

OUTPUT

```
Enter the clipping window limits : Xwmin  10
Enter the clipping window limits : Ywmin  10
Enter the clipping window limits : Xwmax  100
Enter the clipping window limits : Ywmax  100

Enter the end points of the line : P1->x  -20

Enter the end points of the line : P1->y  -30

Enter the end points of the line : P2->x  120

Enter the end points of the line : P2->y  180_
```



LINE CLIPPING - NICHOLL-LEE-NICHOLL

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

int xmin,ymin,xmax,ymax,a,b;

int first_end_point_region(int x,int y);
int findRegionP1(int,int);
void clipline1(int,int,int,int);
void clipline2(int,int,int,int);
void clipline3(int,int,int,int);

void main()
{
 int x1,y1,x2,y2;
 int gdriver = DETECT, gmode;
 int ch;
 float m;
 clrscr();
 cout<<"\nEnter the xmin:->";
 cin>>xmin;
 cout<<"\nEnter the ymin:->";
 cin>>ymin;
 cout<<"\nEnter the xmax:->";
 cin>>xmax;
 cout<<"\nEnter the ymax:->";
 cin>>ymax;
 cout<<"Enter the x1:->";
 cin>>x1;
 cout<<"Enter the y1:->";
 cin>>y1;
 cout<<"Enter the x2:->";
 cin>>x2;
 cout<<"Enter the y2:->";
 cin>>y2;
 initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
 setcolor(12);
 a=getmaxx()/2;
 b=getmaxy()/2;
 line(0,b,2*a,b);
 line(a,0,a,2*b);
 rectangle(a+xmin,b-ymin,a+xmax,b-ymax);
 setcolor(10);
 line(a+x1,b-y1,a+xmin,b-ymin);
 line(a+x1,b-y1,a+xmax,b-ymin);
 line(a+x1,b-y1,a+xmax,b-ymax);
 line(a+x1,b-y1,a+xmin,b-ymax);
 getch();
 setcolor(12);
 line(0,b,2*a,b);
 line(a,0,a,2*b);
```

```
  setcolor(3);
  line(a+x1,b-y1,a+x2,b-y2);
  getch();
  ch=first_end_point_region(x1,y1);
  switch(ch)
  {
    case 1 : clipline1(x1,y1,x2,y2);
                 break;
    case 2 : clipline2(x1,y1,x2,y2);
                 break;
    case 3 : clipline3(x1,y1,x2,y2);
                 break;
    default: cout<<"\nInvalid Input: ";
  };
  getch();
}

int first_end_point_region(int x,int y)
{
  if(x>=xmin && x<=xmax && y>=ymin && y<=ymax)
    return 1;
  else
    if(x<xmin && y>=ymin && y<=ymax)
     return 2;
  else
     if(x<=xmin && y<=ymin)
      return 3;
  else
      return 0;
}

/* point p1 is inside the clip window */
void clipline1(int x1,int y1,int x2,int y2)
{ int draw=1;
  float m,m1,m2,m3,m4;
  int nx1,ny1,nx2,ny2;
  /* calculate slopes for all the lines passing thru vertices
             and including the input line :- */
  m=((float)(y2-y1))/(x2-x1);
  m1=((float)(ymin-y1))/(xmin-x1);
  m2=((float)(ymin-y1))/(xmax-x1);
  m3=((float)(ymax-y1))/(xmax-x1);
  m4=((float)(ymax-y1))/(xmin-x1);
  nx1=x1;
  ny1=y1;
  // point p2 is in "below" region
  if(((abs(m)>=m1 && x2<x1) || (abs(m)>abs(m2) && x2>x1)) && y1>y2)
  { cout<<"working"; getch();
    // point p2 is also inside clip window
    if(y2>ymin)
    {
      nx2=x2;
      ny2=y2;
    }
    // point p2 is outside clip window
    else
```

```
  {
   ny2=ymin;
   nx2=x1+(ymin-y1)/m;
  }
}
// point p2 is on right side of clip window
else if(m>m2 && m<m3 && x2>=x1)
{ // point p2 is inside clip window
 if(x2<xmax)
 {
  nx2=x2;
  ny2=y2;
 }
 // point p2 is outside clip window
 else
 {
  nx2=xmax;
  ny2=y1+(xmax-x1)*m;
 }
}
// point p2 is on bottom side of clip window
else if((abs(m)>=m3 && x2>x1) || (abs(m)>abs(m4) && x2<x1))
{ // point p2 is inside clip window
 if(y2<ymax)
 {
  nx2=x2;
  ny2=y2;
 }
 // point p2 is outside clip window
 else
 {
  ny2=ymax;
  nx2=x1+(ymax-y1)/m;
 }
}
// point p2 is on left side of clip window
else if(m>m4 && m<m1)
{ // point p2 is inside the clip window
 if(x2>xmin)
 {
  nx2=x2;
  ny2=y2;
 }
 // point p2 is outside the clip window
 else
 {
  nx2=xmin;
  ny2=y1+(xmin-x1)*m;
 }
}
getch();
setcolor(12);
rectangle(a+xmin,b-ymin,a+xmax,b-ymax);
if(draw)
{
 setcolor(10);
```

```
   line(a+x1,b-y1,a+xmin,b-ymin);
   line(a+x1,b-y1,a+xmax,b-ymin);
   line(a+x1,b-y1,a+xmax,b-ymax);
   line(a+x1,b-y1,a+xmin,b-ymax);
   setcolor(5);
   line(a+nx1,b-ny1,a+nx2,b-ny2);
  }
}

/* Point p1 is in the edge region */
void clipline2(int x1,int y1,int x2,int y2)
{ int draw=1;
  float m,m1,m2,m3,m4;
  int nx1,ny1,nx2,ny2;
  m=((float)(y2-y1))/(x2-x1);
  m1=((float)(ymin-y1))/(xmin-x1);
  m2=((float)(ymin-y1))/(xmax-x1);
  m3=((float)(ymax-y1))/(xmax-x1);
  m4=((float)(ymax-y1))/(xmin-x1);
  // Point p2 is in Left-bottom region
  if(m>m1 && m<m2 && x2>xmin)
  { // Point p2 is inside the clip window
    if(y2>ymin)
    {
      nx1=xmin;
      ny1=y1+m*(xmin-x1);
      nx2=x2;
      ny2=y2;
    }
    // Point p2 is outside the clip window
    else
    {
      nx1=xmin;
      ny1=y1+m*(xmin-x1);
      ny2=ymin;
      nx2=x1+(ymin-y1)/m;
    }
  }
  // Point p2 is in Left-Right region
  else if(m>m2 && m<m3 && x2>xmin)
  { // Point p2 is inside the clip window
    if(x2<xmax)
    {
      nx1=xmin;
      ny1=y1+m*(xmin-x1);
      nx2=x2;
      ny2=y2;
    }
    // Point p2 is outside the clip window
    else
    {
      nx1=xmin;
      ny1=y1+m*(xmin-x1);
      nx2=xmax;
      ny2=y1+(xmax-x1)*m;
    }
```

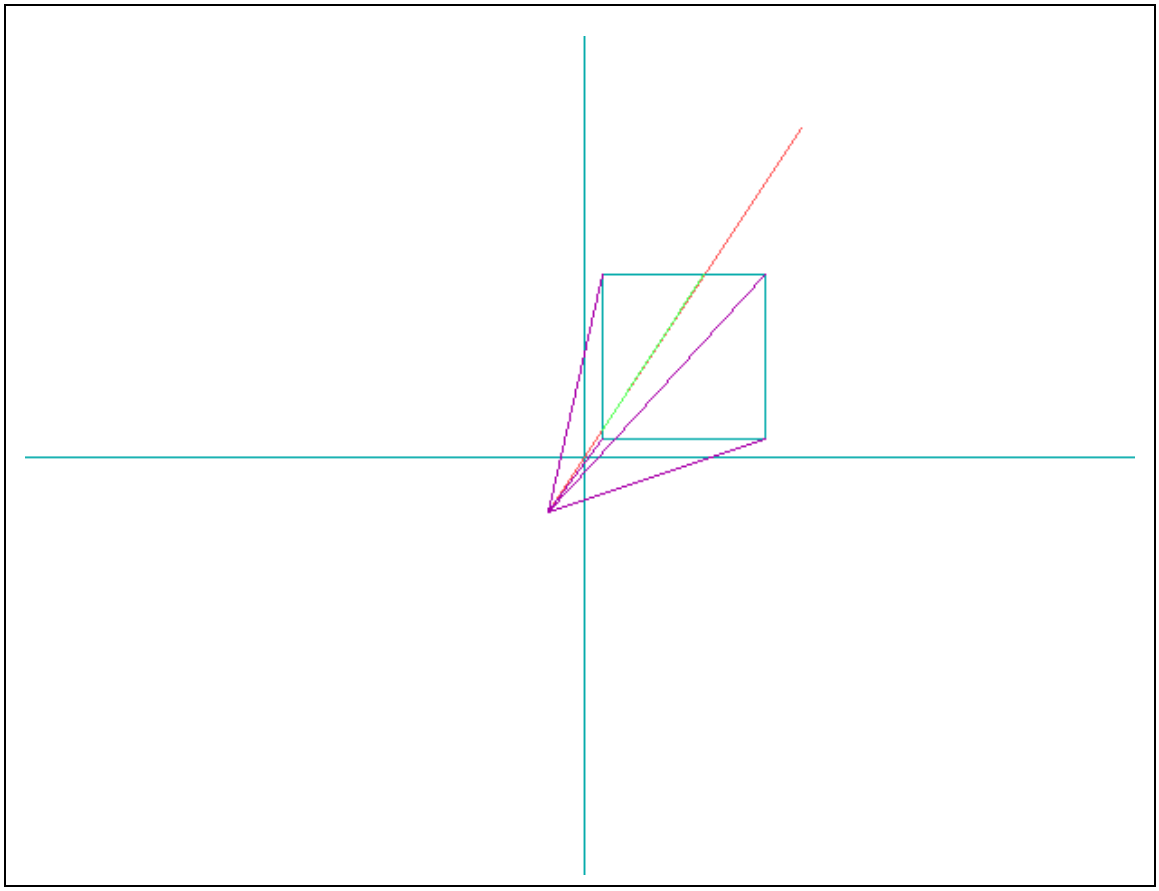
```
 }
 // Point p2 is in Left-top region
 else if(m>m3 && m<m4 && x2>xmin)
 { // Point p2 is inside the clip window
  if(y2<ymax)
  {
   nx1=xmin;
   ny1=y1+m*(xmin-x1);
   nx2=x2;
   ny2=y2;
  }
  // Point p2 is outside the clip window
  else
  {
   nx1=xmin;
   ny1=y1+m*(xmin-x1);
   ny2=ymax;
   nx2=x1+(ymax-y1)/m;
  }
 }
 else
  draw=0;
 setcolor(12);
 rectangle(a+xmin,b-ymin,a+xmax,b-ymax);
 if(draw)
 {
  setcolor(10);
  line(a+x1,b-y1,a+xmin,b-ymin);
  line(a+x1,b-y1,a+xmax,b-ymin);
  line(a+x1,b-y1,a+xmax,b-ymax);
  line(a+x1,b-y1,a+xmin,b-ymax);
  setcolor(5);
  line(a+nx1,b-ny1,a+nx2,b-ny2);
 }
}

/* Point p1 is in the Corner Region */
void clipline3(int x1,int y1,int x2,int y2)
{
 int draw=1;
 float m,m1,m2,m3,m4,tm1,tm2;
 int nx1,ny1,nx2,ny2;
 int flag,t;
 tm1=((float)(ymin-y1))/(xmin-x1);
 tm2=((float)(ymax-ymin))/(xmax-xmin); //diagonal slope
 m=((float)(y2-y1))/(x2-x1);
 m1=((float)(ymin-y1))/(xmax-x1);
 m2=((float)(ymax-y1))/(xmax-x1);
 m3=((float)(ymin-y1))/(xmin-x1);
 m4=((float)(ymax-y1))/(xmin-x1);
 // Point p1 is towards the left side of the clip window (case2)
 if(tm1<tm2)
 {
  flag=2;
  t=m2;
  m2=m3;
```

```
 m3=t;
}
// Point p1 is towards the top side of the clip window (case1)
else
 flag=1;

// Point p2 is in the bottom-Right region
if(m>m1 && m<m2)
{
 // Point p2 is outside the clip window
 if(x2>xmax && y2>ymin)
 {
  ny1=ymin;
  nx1=x1+(ymin-y1)/m;
  nx2=xmax;
  ny2=y1+m*(xmax-x1);
 }
 // Point p2 is inside the clip window
 else if(y2>ymin && x2<xmax)
 {
  ny1=ymin;
  nx1=x1+(ymin-y1)/m;
  ny2=y2;
  nx2=x2;
 }
}
// Point p2 is Left-Right or Top-Bottom region
else if(m>m2 && m<m3)
{
  // Point p2 is in Top-Bottom region (case1)
 if(flag==1)
 {
 // Point p2 is outside the clip window
  if(y2>=ymax)
  {
        ny1=ymin;
        nx1=x1+(ymin-y1)/m;
        nx2=x1+(ymax-y1)/m;
        ny2=ymax;
  }
  // Point p2 is inside the clip window
  else if(y2>=ymin)
  {
   ny1=ymin;
   nx1=x1+(ymin-y1)/m;
   nx2=x2;
   ny2=y2;
  }
 }
 // Point p2 is in Left-Right region (case2)
 else
 {
  // Point p2 is outside the clip window
  if(x2>=xmax)
  {
        nx1=xmin;
```

```
            ny1=y1+m*(xmin-x1);
            nx2=xmax;
            ny2=y1+m*(xmax-x1);
      }
      // Point p2 is inside the clip window
      else if(x2>=xmin)
      {
             nx1=xmin;
             ny1=y1+m*(xmin-x1);
             nx2=x2;
             ny2=y2;
       }
     }
    }
    // Point p2 is in Left-top region
    else if(m>m3 && m<m4)
    {
     // Point p2 is outside the clip window
     if(y2>=ymax)
     {
      nx1=xmin;
      ny1=y1+m*(xmin-x1);
      nx2=x1+(ymax-y1)/m;
      ny2=ymax;
     }
     // Point p2 is inside the clip window
     else if(y2>=ymin)
     {
      nx1=xmin;
      ny1=y1+m*(xmin-x1);
      ny2=y2;
      nx2=x2;
     }
    }
    else
     draw=0;
    getch();
    setcolor(12);
    rectangle(a+xmin,b-ymin,a+xmax,b-ymax);
    if(draw)
    {
     setcolor(10);
     line(a+x1,b-y1,a+xmin,b-ymin);
     line(a+x1,b-y1,a+xmax,b-ymin);
     line(a+x1,b-y1,a+xmax,b-ymax);
     line(a+x1,b-y1,a+xmin,b-ymax);
     setcolor(5);
     line(a+nx1,b-ny1,a+nx2,b-ny2);
    }
}
```

OUTPUT

```
Enter the xmin:->10

Enter the ymin:->10

Enter the xmax:->100

Enter the ymax:->100
Enter the x1:->-20
Enter the y1:->-30
Enter the x2:->120
Enter the y2:->180_
```



POLYGON CLIPPING - SUTHERLAND HODGEMANN

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

struct point
{
 int x,y;
};

int createlist(int i, point cl[], int nc, point s[], int ns)
{
 point t[20];
 int k=0;

 if(i==0)    // TOP EDGE
 {
  for(int j=0;j<ns;j++)
  {
   // o -> i
   if(s[j].y<cl[0].y && s[j+1].y>cl[0].y)
   {
        //find point of intersection
        int ax = int(((cl[0].y-s[j].y)*(s[j+1].x-s[j].x)/(s[j+1].y-s[j].y)*1.0)+s[j].x);
        t[k].x = ax;
        t[k].y = cl[0].y;
        k++;
        t[k] = s[j+1];
        k++;
   }
   //i -> o
   else if(s[j].y>cl[0].y && s[j+1].y<cl[0].y)
   {
        int ax = int(((cl[0].y-s[j].y)*(s[j+1].x-s[j].x)/(s[j+1].y-s[j].y)*1.0)+s[j].x);
        t[k].x = ax;
        t[k].y = cl[0].y;
        k++;
   }
   //i -> i
   else if(s[j].y>cl[0].y && s[j+1].y>cl[0].y)
   {
        t[k] = s[j+1];
        k++;
   }
   //o -> o => do nothing
  }
 }
 else if(i==1)         // RIGHT EDGE
 {
  for(int j=0;j<ns;j++)
  {
   // o -> i
   if(s[j].x>cl[1].x && s[j+1].x<cl[1].x)
```

```
    {
        //find point of intersection
        int ay = int(((cl[1].x-s[j].x)*(s[j+1].y-s[j].y)/(s[j+1].x-s[j].x)*1.0)+s[j].y);
        t[k].x = cl[1].x;
        t[k].y = ay;
        k++;
        t[k] = s[j+1];
        k++;
    }
    //i -> o
    else if(s[j].x<cl[1].x && s[j+1].x>cl[1].x)
    {
        int ay = int(((cl[1].x-s[j].x)*(s[j+1].y-s[j].y)/(s[j+1].x-s[j].x)*1.0)+s[j].y);
        t[k].x = cl[1].x;
        t[k].y = ay;
        k++;
    }
    //i -> i
    else if(s[j].x<cl[1].x && s[j+1].x<cl[1].x)
    {
        t[k] = s[j+1];
        k++;
    }
    //o -> o => do nothing
  }
}
else if(i==2)                    //  BOTTOM EDGE
{
  for(int j=0;j<ns;j++)
  {
    // o -> i
    if(s[j].y>cl[2].y && s[j+1].y<cl[2].y)
    {
        //find point of intersection
        int ax = int(((cl[2].y-s[j].y)*(s[j+1].x-s[j].x)/(s[j+1].y-s[j].y)*1.0)+s[j].x);
        t[k].x = ax;
        t[k].y = cl[2].y;
        k++;
        t[k] = s[j+1];
        k++;
    }
    //i -> o
    else if(s[j].y<cl[2].y && s[j+1].y>cl[2].y)
    {
        int ax = int(((cl[2].y-s[j].y)*(s[j+1].x-s[j].x)/(s[j+1].y-s[j].y)*1.0)+s[j].x);
        t[k].x = ax;
        t[k].y = cl[2].y;
        k++;
    }
    //i -> i
    else if(s[j].y<cl[2].y && s[j+1].y<cl[2].y)
    {
        t[k] = s[j+1];
        k++;
    }
    //o -> o => do nothing
```

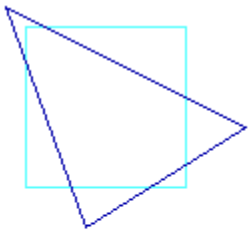
```
  }
 }
 else if(i==3)    //  LEFT EDGE
 {
  for(int j=0;j<ns;j++)
  {
    // o -> i
    if(s[j].x<cl[0].x && s[j+1].x>cl[0].x)
    {
          //find point of intersection
          int ay = int(((cl[0].x-s[j].x)*(s[j+1].y-s[j].y)/(s[j+1].x-s[j].x)*1.0)+s[j].y);
          t[k].x = cl[0].x;
          t[k].y = ay;
          k++;
          t[k] = s[j+1];
          k++;
    }
    //i -> o
    else if(s[j].x>cl[0].x && s[j+1].x<cl[0].x)
    {
          int ay = int(((cl[0].x-s[j].x)*(s[j+1].y-s[j].y)/(s[j+1].x-s[j].x)*1.0)+s[j].y);
          t[k].x = cl[0].x;
          t[k].y = ay;
          k++;
    }
    //i -> i
    else if(s[j].x>cl[0].x && s[j+1].x>cl[0].x)
    {
          t[k] = s[j+1];
          k++;
    }
    //o -> o => do nothing
  }
 }
 t[k]=t[0];

 for(int l=0;l<=k;l++)
 {
  s[l]=t[l];
 }

 return k;
}

/*int is_out(point p, point cl[], int nc)
{

 // 1. I have taken clockwise orientation positive
 // 2. So any point is inside the polygon if it is to the left of every edge
 // 3. Otherwise it is outside.
 // 4. Let edge be p1p2(p1 & p2 in clockwise order). Let point be P.
 //    therefore, if slope(p1p2) > slope(p1P), for every edge,
 //    then the point is inside the polygon


 int in = 0;
```

```
  int out = 0;
  int i=0;
  while (i<nc && out==0)
  {
    dec = ((c[i+1].y - c[i].y) * (p.x - c[i].x)) - ((p.y - c[i].y) * (c[i+1].x - c[i].x));
    if(dec < 0)
      out =1;
    i++;
  }
  return out;
  if((p.x >=  cl[0].x)&&(p.x <= cl[1].x)&&(p.y >= cl[0].y)&&(p.y <= cl[2].y))
    return 0;
  else
    return 1;
}            */

void suthodg( point cl[], int nc, point s[], int ns)
{
  for(int i=0;i<nc;i++)
  {
    ns = createlist(i, cl, nc, s, ns);
  }

  setcolor(GREEN);
  for(i=0;i<ns;i++)
  {
    line(s[i].x,s[i].y,s[i+1].x,s[i+1].y);
  }

}

void main()
{
  clrscr();

  point cl[4];
  point sub[10], dup_sub[10];

  int nc;
  cout<<"    Clipping Polygon ";

  //min is top left and max is bottom right

  cout<<"\n Enter the co-ordinates (x,y) ";
  cout<<"\n Xwmin : ";
  cin>>cl[0].x;
  cout<<"\n Ywmin : ";
  cin>>cl[0].y;
  cout<<"\n Xwmax : ";
  cin>>cl[2].x;
  cout<<"\n Ywmax : ";
  cin>>cl[2].y;
  cl[1].x = cl[2].x;
  cl[1].y = cl[0].y;
  cl[3].x = cl[0].x;
  cl[3].y = cl[2].y;
```

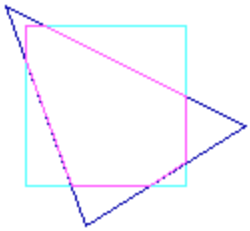
```
  int ns;
  cout<<"    Subject Polygon ";
  do
  {
   cout<<"\n Enter the no. of vertices : ";
   cin>>ns;
  }while(ns>10);
  cout<<"\n Enter the co-ordinates in clockwise order (x,y) ";

  for(int i=0;i<ns;i++)
  {
   cout <<i+1<<". ";
   cin>>sub[i].x>>sub[i].y;
   dup_sub[i] = sub[i];
  }
  sub[i] = sub[0];
  dup_sub[i] = sub[0];
  int gdriver = DETECT, gmode;
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");

  setcolor(RED);
  for(i=0;i<3;i++)
  {
   line(cl[i].x, cl[i].y, cl[i+1].x, cl[i+1].y);
  }
  line( cl[3].x, cl[3].y, cl[0].x, cl[0].y);

  setcolor(YELLOW);
  for(i=0;i<ns;i++)
  {
   line(sub[i].x,sub[i].y,sub[i+1].x,sub[i+1].y);
  }
  getch();
  suthodg(cl,4,dup_sub,ns);

  getch();
  closegraph();
}
```

OUTPUT

```
   Clipping Polygon
 Enter the co-ordinates (x,y)
 Xwmin : 20

 Ywmin : 20

 Xwmax : 100

 Ywmax : 100
   Subject Polygon
 Enter the no. of co-ordinates : 4

 Enter the co-ordinates in clockwise order (x,y) 1. 10 10
2. 70 40
3. 130 70
4. 50 120_
```



Original configuration



Clipped Polygon

POLYGON CLIPPING - WEILER ATHERTON

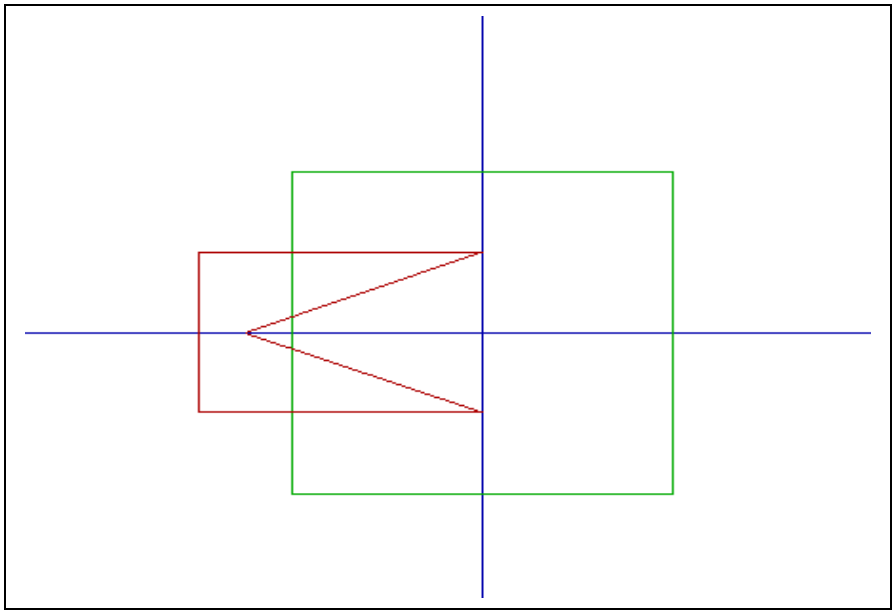
```
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

float sdx[15],sdy[15];
int i,w=0,h;
void sort(float sdy[],int h)
{
 float temp;
 for(int j=0;j<=h-1;j++)
 {
  for(i=0;i<h-1-j;i++)
  {
   if(sdy[i]>sdy[i+1])
   {
        temp=sdy[i];
        sdy[i]=sdy[i+1];
        sdy[i+1]=temp;
   }
  }
 }
}

struct ather
{
 float x;
 float y;
 float io;
 float vis;
};
struct ather z[20];

void main()
{
 int gd=DETECT;
 int gm;
 initgraph(&gd,&gm,"c:\\tc\\bgi");
 int n,m,s;
 float px[15]={0};
 float py[15]={0};
 float pdx[15],pdy[10];
 float outx[15]={0};
 float outy[15]={0};
 float xmin,ymin,xmax,ymax;
 printf("enter xmin,ymin,xmax,ymax");
 scanf("%f%f%f%f",&xmin,&ymin,&xmax,&ymax);
 rectangle(320+xmin,240-ymax,320+xmax,240-ymin);
 printf("enter the no. of vertices (n)");
 scanf("%d",&n);
 printf("enter the x coordinate of all vertices");
```

```
for(m=0;m<n;m++)
{
 scanf("%f",&px[m]);
}

printf("enter the y coordinate of all vertices");
for(m=0;m<n;m++)
{
 scanf("%f",&py[m]);
}
rectangle(320+xmin,240-ymax,320+xmax,240-ymin);
px[n]=px[0];py[n]=py[0];
for(s=0;s<n;s++)
{
 line(320+px[s],240-py[s],320+px[s+1],240-py[s+1]);
}
getch();
px[n]=px[0];
py[n]=py[0];  int l=0;
for(m=0;m<n;m++)
{
 if(px[m]>=xmin && px[m+1]<=xmin)
 {
  pdx[m]=xmin;
  pdy[m]=py[m]+((py[m+1]-py[m])/(px[m+1]-px[m]))*(xmin-px[m]);
  outx[l]=pdx[m];outy[l]=pdy[m];
  z[l].io=1;
  l++;
 }
 if(px[m]>=xmin && px[m+1]>=xmin)
 {
  outx[l]=px[m+1];outy[l]=py[m+1];
  z[l].io=0;
  l++;
 }
 if(px[m]<=xmin && px[m+1]>=xmin)
 {
  pdx[m]=xmin;
  pdy[m]=py[m]+((py[m+1]-py[m])/(px[m+1]-px[m]))*(xmin-px[m]);
  outx[l]=pdx[m];outy[l]=pdy[m];
  z[l].io=0;
  l++;
  outx[l]=px[m+1];outy[l]=py[m+1];
  z[l].io=0;
  l++;
 }
}
outx[l]=outx[0];outy[l]=outy[0];
setcolor(GREEN);
for(i=0;i<l;i++)
{
 if(outx[i]==xmin)
 {
  sdx[w]=outx[i];
  sdy[w]=outy[i];
  w++;
```

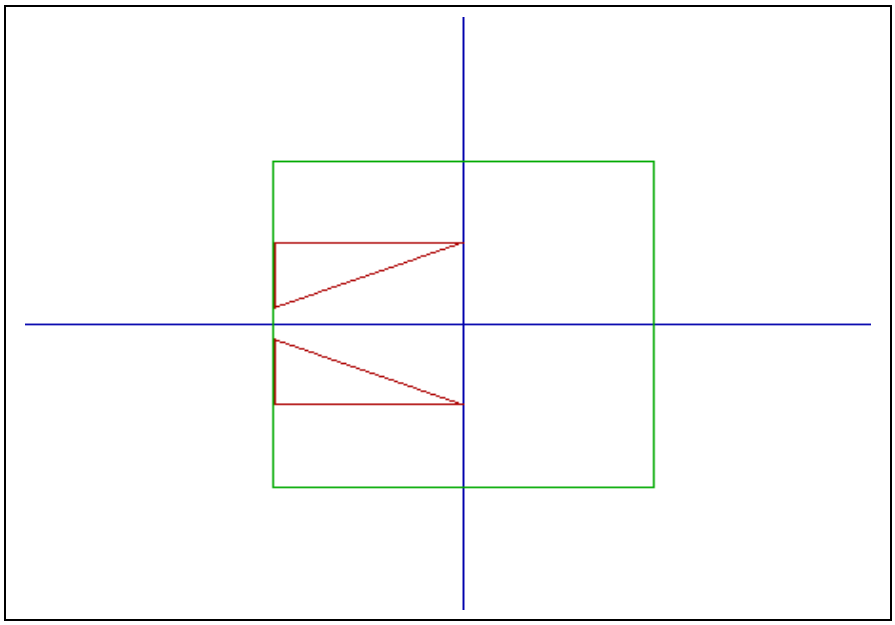
```
  }
}
sort(sdy,w);
outx[l]=outx[0];outy[l]=outy[0];
for(i=0;i<=l;i++)
{
  z[i].x=outx[i];
  z[i].y=outy[i];
  z[i].vis=0;
}
s=0;
for(m=0;m<=l-1;m++)
{
  outx[l]=outx[0];outy[l]=outy[0];
  sdx[w+1]=sdx[0];sdy[w+1]=sdy[0];
  if(z[s].io==0)
  {
    line(320+outx[s],240-outy[s],320+outx[s+1],240-outy[s+1]);
    z[s].vis=1;
    z[s+l].vis=1;
  }
  else if(z[s].io==1)
  {
    for(i=0;i<=w;i++)
    {
          if(sdy[i]==outy[s])
          {
            line(320+sdx[i],240-sdy[i],320+sdx[i+1],240-sdy[i+1]);
            z[s].vis=1;
            z[s+l].vis=1;
            break;
          }
    }
    for(int j=0;j<l;j++)
    {
          if(sdy[i+1]==z[j].y)
          {
            s=j;
            line(320+outx[s],240-outy[s],320+outx[s+1],240-outy[s+1]);
            z[s].vis=1;
            z[s+l].vis=1;
            break;
          }
    }
  }
  if(s<=l-1)
  {
    s++;
  }
  else
  {
    s=0;
  }
  if(s==l)
  {
    s=0;
```

```
    }
    int p=s;

    while(z[s].vis == 1)
    {
      s++;
      if(s==p+l)
      {
            break;
      }
    }
  }
  getch();
}
```

OUTPUT



Original Configuration



Clipped Polygon

SEED FILL

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

int main(void)
{
 int gdriver = DETECT, gmode, errorcode;
 int i,n,c,boun;
 float x[20],y[20],a,b,l,m;
 int polyfill(float,float,int,int);
 int plyfill(float,float,int,int);
 int plyfll(float,float,int,int);
 int plyfil(float,float,int,int);
 initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
 cout<<"enter the no of vertices : ";
 cin>>n;
 for(i=0;i<n;i++)
 {
  cout<<"enter the coordinates : ";
  cin>>x[i]>>y[i];
 }
 cout<<"enter the color code : ";
 cin>>c;
 cout<<"enter the boundry color : ";
 cin>>boun;
 cout<<"enter an interior pt : ";
 cin>>a>>b;
 x[i]=x[0],y[i]=y[0];
 setcolor(boun);
 for(i=0;i<n;i++)
  line(320+x[i],240-y[i],320+x[i+1],240-y[i+1]);
 setcolor(WHITE);
 l=a,m=b+1;
 getch();
 polyfill(a,b,c,boun);
 plyfll(a-1,b,c,boun);
 plyfil(l,m,c,boun);
 plyfill(l+1,m,c,boun);
 getch();
}

int polyfill(float p,float q,int c,int bo)
{
 int r;
 r=getpixel(320+p,240-q);
 if((r!=bo)&&(r!=c))
 {
  putpixel(320+p,240-q,c);
  polyfill(p,(q-1),c,bo);
  polyfill(p+1,q,c,bo);
 }
}
```

```
int plyfill(float p,float q,int c,int bo)
{
 int r;
 r=getpixel(320+p,240-(q));
 if((r!=bo)&&(r!=c))
 {
  putpixel(320+p,240-q,c);
  plyfill(p,(q+1),c,bo);
  plyfill(p+1,q,c,bo);
 }
}

int plyfil(float p,float q,int c,int bo)
{
 int r;
 r=getpixel(320+p,240-(q));
 if((r!=bo)&&(r!=c))
 {
  putpixel(320+p,240-q,c);
  plyfil(p,(q+1),c,bo);
  plyfil(p-1,q,c,bo);
 }
}

int plyfll(float p,float q,int c,int bo)
{
 int r;
 r=getpixel(320+p,240-(q));
 if((r!=bo)&&(r!=c))
 {
  putpixel(320+p,240-q,c);
  plyfll(p,(q-1),c,bo);
  plyfll(p-1,q,c,bo);
 }
}
```

OUTPUT

```
enter the no of vertices 5
enter the coordinates : 10 10
enter the coordinates : 30 10
enter the coordinates : 40 20
enter the coordinates : 30 30
enter the coordinates : 10 30
enter the color code 4
enter the boundry color 6
enter an interior pt 24 20
```

POLYGON FILLING USING SCANLINE

```
/* scan line approach for filling polygons with horizontal edges.
  (i.e. a general program)
*/
#include<fstream.h>
#include<graphics.h>
#include<conio.h>
#include<alloc.h> //for function free
#include<process.h>//for exit function
// upper limit for no of vertices
const int MAX=20;
//const int BKCOLOR=LIGHTGRAY;
const int BKCOLOR=BLACK;
const int DRCOLOR=RED;
const int FILLCOLOR=WHITE;
const int SPECIAL=BLUE;
const int xorigin=320;
const int yorigin=240;

struct point
{
        int x,y;
};
/*function:gets the vertices of the poly from the suitable source
  (currently using brute force for testing and simplifying purposes)
  paramaters: p1 has the poly, n has no of vertices
*/ void getdata(point p1[MAX],int &n);

/* used to copy p1 to p2 before sorting. as p2=p1 cant be done
*/ void copy(point p2[MAX],point p1[MAX],int n);

/* function:sorts the vertices of the polygon according to increasing y values
   sorting technique: insertion sort
   parameters: p (the polygon),n (used for sorting)
*/ void sort(point p[MAX],int n);

/* for testing purposes
*/ void display(point p[MAX],int n);

struct node
{
        int ymax;
        float x,delx;
        node* link;
};
//sets up a node for the linked lists
node* getnode(int ymax1,float x1,float delx1);

/* GET tasks required:
1. insertion:insertion at end will do as its beneficial for the
   derived class AET.
2. sorting:the list has to be sorted according to the increasing x values
3. access head:give the address stored in the head node for merging the
```

```
  list in AET
5. searching:for constructing the table i need to see if an edge has
  been already inserted or not
6. display:this is required for the testing purposes
*/
class linked_list_GET
{
        protected:
                node* head;
        public:
                linked_list_GET()
                { head=NULL; }

                /* i am passing a node to insert functions as
                  to insert in AET i will remove nodes from GET and put
                  in AET. as they wont be required by GET anymore.
                  for insertion in GET prepare the node by calling getnode
                  function and then call insert.
                  this way i can insert lists at the end and in the beginning.
                */ void insert_at_head(node* n);
                  void insert_at_end(node* n);

                int sort();

                /* for merging a list in the AET address in the head pointer
                  is required.this function returns that address
                */ node* access_head()
                { return head; }

                /* searches the item in the list. in the end child
                  points to the element searching for and parent
                  points to the previous element.
                  here the task is bascially whether an edge has been
                  included in GET or not
                */ int search(node* item,node* &parent,node* &child);

                void display();
};
/* AET tasks required:
1. insertion:have to insert a whole list from GET, hence doing insertion
  at end.(derived from the base class)
2. sorting:the list has to be sorted according to increasing x values
  (derived from the base class)
3. update:as y is increased new intersection points have to be calculated
4. delete:have to delete those edges which are complete
5. drawing:list has to be drawn in pairs
6. display:required for testing purposes (derived from the base class)
*/
class linked_list_AET:public linked_list_GET
{
        public:
                linked_list_AET():linked_list_GET()
                {

                }
                /* for a new scanline the new intersection points have
```

```
                    to be calculated. this is done by x=x+delx (coherence
                    property). update function performs this task
                  */ void update();

                  /* this function checks whether the first or the last
                    element has to be deleted or not
                    return values:
                    0:no
                    1:yes
                  */ int del_first_or_last(int y);

                  /* searches the item in the list. in the end child
                    points to the element searching for and parent
                    points to the previous element.
                    return values:
                    not found:0
                    found:1
                  */ int search(int y,node* &parent,node* &child);

                  /* return values:
                    no deletion:0
                    deletion:1
                  */ int delete_first();

                  /* have to use this only when in AET ymax of an edge
                    is reached. hence search for y and delete if found
                    also i want to know if anyone of the first or the last
                    node is being deleted.
                    return values:
                    no deletion:0
                    deletion:1
                  */ int delete_item(int y);

                  /* to fill the polygon we have to draw in pairs from the
                    x values of the node at the y value passed as an argument.
                  */ void draw(int y);
};

/* array for the lists of GET. as no of edges cant exceed no
   of vertices,no of lists are even lesser so this is a safe upper limit
   the array yindex stores the y value at which the list at the corresponding
   index in both arrays would be in the actual algo.
*/ linked_list_GET GET[MAX];
   int yindex[MAX]={0};

/* for non horizontal edges gets a node and inserts it in appropriate list
   paramenters:
   a: the current vertex
   b: the adjoining vertex
*/ void make_insert_edge(point a,point b,int ycur);

/* take each edge one by one starting from ymin and put it in the
   corredponding position in GET and the y value in yindex
   paramenters: p1 ,p2, n (polygon data)
   after this function call GET is completely ready
*/ void make_GET(point p1[MAX],point p2[MAX],int n);
```

```
void goto_graphics_mode();
void draw_poly(point p[MAX],int n);

/* start with empty list and move from ymin to ymax.
  for every y do
            update
            merge
            sort
            delete
            draw
  if delete from 1st or last then first draw and delete,
  then do the normal draw
  parameters:
  p2,n:required to get ymin and ymax
*/ void process_AET(point p2[MAX],int n);

void scanline(point p1[MAX],point p2[MAX],int n)
{
          /* polygon obtained. now sort them according to increasing y values
            and store them in another array
          */ copy(p2,p1,n);
            sort(p2,n);

          /* polygon in both arrays. make GET(global edge table).
            this completes the GET with all the lists sorted also
          */ make_GET(p1,p2,n);

          goto_graphics_mode();
          draw_poly(p1,n);

          /* now just process AET. this fills the polygon
          */ process_AET(p2,n);
}
void main()
{

          /*p1 is the polygon with the vertices in the cyclic order
            n has the no of vertices in the polygon
          */
          point p1[MAX],p2[MAX];
          int n;
          /*the vertices of the polygon should be stored in the text file
          polygon.txt but for testing and simplifying purposes using
          brute force, so first retrieve them and store in p1
          */ getdata(p1,n);

          scanline(p1,p2,n);

          getch();
}

void getdata(point p1[MAX],int &n)
{
          n=19;
```

```
	p1[0].x=2; p1[0].y=5;
	p1[1].x=2; p1[1].y=7;
	p1[2].x=4; p1[2].y=7;
	p1[3].x=4; p1[3].y=9;

	p1[4].x=6; p1[4].y=9;
	p1[5].x=6; p1[5].y=7;
	p1[6].x=8; p1[6].y=7;
	p1[7].x=9; p1[7].y=9;

	p1[8].x=10; p1[8].y=9;
	p1[9].x=10; p1[9].y=7;
	p1[10].x=14; p1[10].y=7;
	p1[11].x=12; p1[11].y=1;

	p1[12].x=10; p1[12].y=1;
	p1[13].x=10; p1[13].y=3;
	p1[14].x=8; p1[14].y=3;
	p1[15].x=8; p1[15].y=1;

	p1[16].x=6; p1[16].y=1;
	p1[17].x=6; p1[17].y=3;
	p1[18].x=4; p1[18].y=3;
	for(int i=0;i<n;i++)
	{
		p1[i].x*=10;
		p1[i].y*=10;
	}

}
void copy(point p2[MAX],point p1[MAX],int n)
{
	for(int i=0;i<n;i++)
	{
		p2[i].x=p1[i].x;
		p2[i].y=p1[i].y;
	}
}
void sort(point p[MAX],int n)
{
	//using insertion sort
	point temp;
	for(int i=1;i<n;i++)
	{
		temp=p[i];
		for(int j=i; ( (temp.y < p[j-1].y) && j>0) ;j--)
			p[j]=p[j-1];
		p[j]=temp;
	}
}
void display(point p[MAX],int n)
{
	for(int i=0;i<n;i++)
	{
		cout<<p[i].x<<","<<p[i].y<<"     ";
	}
```

```
        getch();
}
node* getnode(int ymax1,float x1,float delx1)
{
        node* n=new node;
        n->ymax=ymax1;
        n->x=x1;
        n->delx=delx1;
        n->link=NULL;
        return n;
}
void linked_list_GET::insert_at_head(node* n)
{
        //this enables a list to be inserted at head
        node* temp=n;
        while(temp->link!=NULL)
                temp=temp->link;
        temp->link=head;
        head=n;
}
void linked_list_GET::insert_at_end(node* n)
{
        if(head==NULL)
                insert_at_head(n);
        else
        {
                node* temp=head;
                //go to the end
                while(temp->link!=NULL)
                        temp=temp->link;
                /* place the new data at the end
                  more than one node can be inserted at a time.
                  to be specific a list can be concatenated at the end
                */
                temp->link=n;
        }
}
int linked_list_GET::sort()
{
        if(head==NULL)
                return 0;
        else
        {
                node* temp1=head;
                node* temp2=head;
                node* temp_pos=NULL;
                int temp_ymax;
                float temp;
                while(temp1->link!=NULL)
                        temp1=temp1->link;
                //temp1 is at the last node
                while(temp1!=head)
                {
                        temp2=head;
                        while(temp2!=temp1)
                        {
```

```
                                        if( temp2->x > temp2->link->x )
                                        {
                                                //swapping values
                                                temp_ymax=temp2->ymax;
                                                temp2->ymax=temp2->link->ymax;
                                                temp2->link->ymax=temp_ymax;

                                                temp=temp2->x;
                                                temp2->x=temp2->link->x;
                                                temp2->link->x=temp;

                                                temp=temp2->delx;
                                                temp2->delx=temp2->link->delx;
                                                temp2->link->delx=temp;
                                        }
                                        temp_pos=temp2;
                                        temp2=temp2->link;
                                }
                                temp1=temp_pos;
                        }
                        return 1;
                }
}
int linked_list_GET::search(node* item,node* &parent,node* &child)
{
        //parent follows the child
        child=head;
        while(child!=NULL)
        {
                if(child->ymax==item->ymax && child->x==item->x && child->delx==item->delx)
                        return 1;
                parent=child;
                child=child->link;
        }
        return 0;
}
void linked_list_GET::display()
{
        if(head==NULL)
                cout<<"Empty list.";
        else
        {
                cout<<"The list is:"<<endl;
                node* temp=head;
                while(temp!=NULL)
                {
                        cout<<temp->ymax<<","<<temp->x<<","<<temp->delx<<"    ";
                        temp=temp->link;
                }
        }
}
void linked_list_AET::update()
{
        node* temp=head;
        //for all nodes do x+=delx
        while(temp!=NULL)
```

```
		{
			(temp->x)=(temp->x)+(temp->delx);
			temp=temp->link;
		}
}
int linked_list_AET::del_first_or_last(int y)
{
		//first element
		if(head->ymax==y)
			return 1;
		else
		{
			node* temp=head;
			//last element
			while(temp->link!=NULL)
				temp=temp->link;
			if(temp->ymax==y)
				return 1;
		}
		return 0;
}
int linked_list_AET::search(int y,node* &parent,node* &child)
{
		//parent follows the child
		child=head;
		while(child!=NULL)
		{
			if(child->ymax==y)
				return 1;
			parent=child;
			child=child->link;
		}
		return 0;
}
int linked_list_AET::delete_first()
{
		if(head==NULL)
			return 0;
		node* n=head;
		head=head->link;
		free(n);
		return 1;
}
int linked_list_AET::delete_item(int y)
{
		node* parent;
		node* child;
		if( search(y,parent,child) )
		{
			if(child==head)
				return	delete_first();
			else
			{
				parent->link=child->link;
				free(child);
				return 1;
```

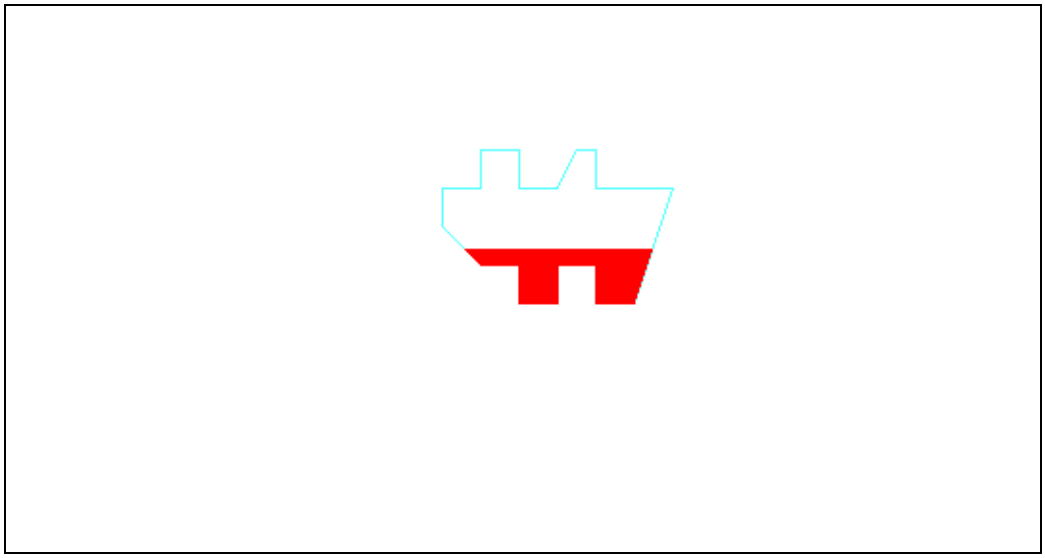
```
                }
        }
        else
                return 0;
}
void linked_list_AET::draw(int y)
{
        node* temp=head;
        int first,next;
        while(temp!=NULL)
        {
                //move forward by 2 to draw in pairs
                first=temp->x;
                temp=temp->link;
                if(temp==NULL)
                        break;
                next=temp->x;
                temp=temp->link;

                line(xorigin+first,yorigin-y,xorigin+next,yorigin-y);

        }
}
void make_insert_edge(point a,point b,int ycur)
{
        if(a.y!=b.y)//for non horizontal edges
        {
                int ymax,x;
                //select ymax and x(ymin)
                if(a.y>b.y)
                {
                        ymax=a.y;
                        x=b.x;
                }
                else
                {
                        ymax=b.y;
                        x=a.x;
                }
                float delx=(a.x-b.x)*1.0/(a.y-b.y);
                node *temp=getnode(ymax,x,delx);

                //check if this edge has already been included in GET or not
                int found=0;
                for(int i=0;i<=ycur;i++)
                {
                        if(GET[i].search(temp,NULL,NULL))
                        {
                                found=1;
                                break;
                        }
                }
                //if edge is not included then include it else free temp
                if(!found)
                        GET[ycur].insert_at_end(temp);
                else
```

```
                        free(temp);
        }
}
void make_GET(point p1[MAX],point p2[MAX],int n)
{
        /* this tells the current y location and is required to know if the
          new edge will be inserted in the current list or next higher list
        */ int ycur=0;

        /*for a vertex these tell the index of adjoining vertices in p1
        */ int e1,e2;

        /* for each vertex in p2 look what edges can be formed leaving aside
          horizontal edges
        */
        yindex[0]=p2[0].y; //otherwise the first pointer will almost always be NULL
        for(int i=0;i<n;i++)
        {
                /* if edges goes in next list increment ycur */
                if( yindex[ycur] < p2[i].y )
                {
                        //list finish so sort it and start a new list
                        GET[ycur].sort();
                        ycur++;
                }
                yindex[ycur]=p2[i].y;

                //search for curent vertex of p2 in p1
                for(int j=0;j<n;j++)
                {
                        if(p1[j].x==p2[i].x && p1[j].y==p2[i].y)
                                break;
                }
                /* a%n=(a%n+n)%n this gives the right result for both
                  positive and negative nos in the required form ie
                  positive no suitable for getting edges from arrays
                */
                e1=((j+1)%n+n)%n;
                e2=((j-1)%n+n)%n;

                node *temp;
                make_insert_edge(p2[i],p1[e1],ycur);
                make_insert_edge(p2[i],p1[e2],ycur);
        }//for
}
void goto_graphics_mode()
{
        int gdriver = DETECT, gmode, errorcode;
        initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
        errorcode = graphresult();
        if (errorcode != grOk)
        {
                cout<<"Graphics error:"<< grapherrormsg(errorcode);
                cout<<"Press any key to halt:";
                getch();
                exit(1);    /* terminate with an error code */
```

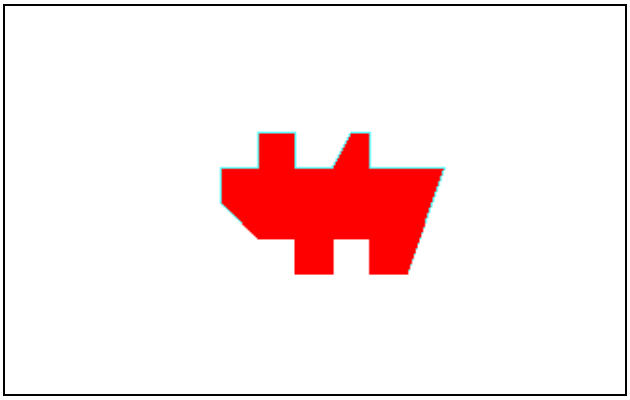
```
        }
        setbkcolor(BKCOLOR);
        setcolor(DRCOLOR);
}

void draw_poly(point p[MAX],int n)
{
        for(int i=0;i<n;i++)
        {
                line(xorigin+p[i].x,yorigin-p[i].y,xorigin+p[(i+1)%n].x,yorigin-p[(i+1)%n].y);
        }
}
void process_AET(point p2[MAX],int n)
{
        linked_list_AET AET;
        /* get ymin and ymax
          ymin=p2[0].y; ymax=p2[n-1].y;
        */
        int ycur=0;
        for(int i=p2[0].y;i<=p2[n-1].y;i++)
        {
                AET.update();

                /* merging
                  if GET list for current i exists merge the list into AET
                  and increment the pointer,for next non empty list, ycur
                */
                if(i==yindex[ycur])
                {
                        AET.insert_at_end( GET[ycur].access_head() );
                        ycur++;
                }

                AET.sort();

                /* deletion
                  if 1st or last element is to be deleted,
                  then first draw and then delete
                  and do this for all the nodes that are to be deleted
                */
                int flag=0;
                /* find if there is deletion from 1st or last node
                  if yes then draw and del all else just del all
                */
                flag=AET.del_first_or_last(i);
                if(flag)
                {
                        setcolor(SPECIAL);
                        AET.draw(i);
                        //AET.display();
                        getch();
                }

                flag=1;
                while(flag)
                {
```

```
                        flag=AET.delete_item(i);
                }
/*              AET.display();
                cout<<endl<<endl;
*/
                setcolor(FILLCOLOR);
                AET.draw(i);
                getch();
        }
}
```

OUTPUT



Intermediate step



Final State

2D TRANSFORMATION

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

int sq[4][2]={{0,0},{100,0},{100,100},{0,100}};

void put_line(int x1,int y1,int x2, int y2)
{
 line(x1+320,250-y1,x2+320,250-y2);
}

void put_box(int sq[4][2])
{
 setcolor(RED);
 line(0,250,640,250);
 setcolor(BLUE);
 line(320,0,320,500);
 setcolor(WHITE);
 for(int i=0;i<3;i++)
 {
  put_line(sq[i][0],sq[i][1],sq[i+1][0],sq[i+1][1]);
 }
 put_line(sq[0][0],sq[0][1],sq[3][0],sq[3][1]);
}

void incr(int sq[4][2], int xincr, int yincr)
{
 for(int i=0;i<4;i++)
 {
  sq[i][0]=sq[i][0]+xincr;
  sq[i][1]=sq[i][1]+yincr;
 }
}

void initia(int sq[4][2])
{
 int gdriver = DETECT, gmode;
 initgraph(&gdriver,&gmode, "c:\\tc\\bgi");
 setcolor(RED);
 line(0,250,640,250);
 setcolor(BLUE);
 line(320,0,320,500);
 setcolor(WHITE);
 put_box(sq);
}

void rot(int sq[4][2], int sq2[4][2], int h, int k, float d)
{
 for(int i=0;i<4;i++)
 {
  sq2[i][0]=(sq[i][0]-h)*cos(d) + (k-sq[i][1])*sin(d)+h;
  sq2[i][1]=(sq[i][0]-h)*sin(d) + (sq[i][1]-k)*cos(d)+k;
```

```
  }
  put_box(sq2);
}

void rot_gen()
{
  char key;
  int sq2[4][2];
  initia(sq);
  float d=0;
  while((key=getch())!='e')
  {
    if(((int)key)==75)
    {
      d+=0.01;
      cleardevice();
      rot(sq,sq2,0,0,d);
    }
    else if(((int)key)==77)
    {
      d-=0.01;
      cleardevice();
      rot(sq,sq2,0,0,d);
    }
  }
  for(int i=0;i<4;i++)
  {
    sq[i][0]=sq2[i][0];
    sq[i][1]=sq2[i][1];
  }
  closegraph();
}

void rot_pt()
{
  char key;
  int sq2[4][2];
  initia(sq);
  float d=0;
  gotoxy(2,2);
  cout<<"Enter point about which rotation is to be done : (x,y) ";
  int rx,ry;
  cin>>rx>>ry;
  while((key=getch())!='e')
  {
    if(((int)key)==75)
    {
      d+=0.01;
      cleardevice();
      rot(sq,sq2,rx,ry,d);
    }
    else if(((int)key)==77)
    {
      d-=0.01;
      cleardevice();
      rot(sq,sq2,rx,ry,d);
```

```
    }
   }
   for(int i=0;i<4;i++)
   {
    sq[i][0]=sq2[i][0];
    sq[i][1]=sq2[i][1];
   }
   closegraph();
}

void translation()
{
  initia(sq);
  int xincr, yincr;
  char key;
  while((key=getch())!='e')
  {
    if(((int)key)==72)
    {
      yincr = 5;
      xincr = 0;
      cleardevice();
      incr(sq,xincr,yincr);
      put_box(sq);
    }
    else if(((int)key)==80)
    {
      yincr=-5;
      xincr=0;
      cleardevice();
      incr(sq,xincr,yincr);
      put_box(sq);
    }
    else if(((int)key)==75)
    {
      yincr=0;
      xincr=-5;
      cleardevice();
      incr(sq,xincr,yincr);
      put_box(sq);
    }
    else if(((int)key)==77)
    {
      yincr=0;
      xincr=5;
      cleardevice();
      incr(sq,xincr,yincr);
      put_box(sq);
    }
  }
  closegraph();
}

void sca(int sq[4][2], float scx, float scy, int h, int k)
{
  for(int i=0;i<4;i++)
```

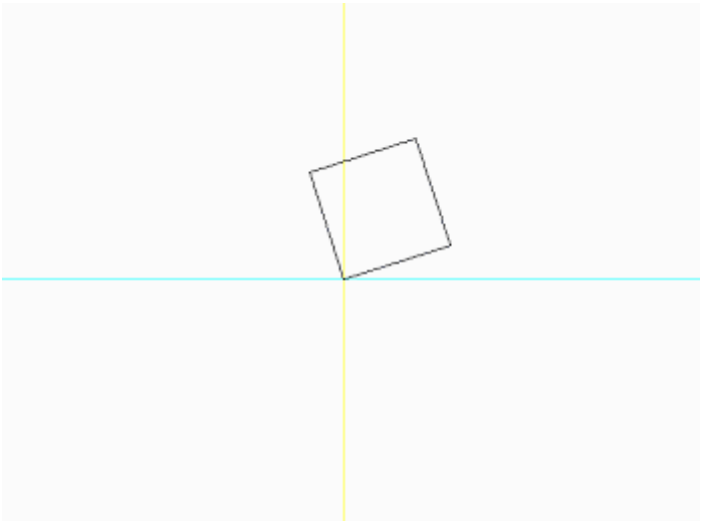
```
  {
   sq[i][0]= sq[i][0]*scx - (scx*h) + h;
   sq[i][1]= sq[i][1]*scy - (scy*k) + k;
  }
  put_box(sq);
}

void scale_fix()
{
  char key;
  int sq2[4][2];
  initia(sq);
  float scx,scy;
  gotoxy(2,2);
  cout<<"Enter fixed point : ";
  int sx,sy;
  gotoxy(3,2);
  cin>>sx>>sy;
  while((key=getch())!='e')
  {
   if(((int)key)==72)
   {
    scy=1.1;
    scx=1.0;
    cleardevice();
    sca(sq,scx,scy,sx,sy);
   }
   else if(((int)key)==80)
   {
    scy=1.0/1.1;
    scx=1.0;
    cleardevice();
    sca(sq,scx,scy,sx,sy);
   }
   else if(((int)key)==75)
   {
    scx=1.0/1.1;
    scy=1.0;
    cleardevice();
    sca(sq,scx,scy,sx,sy);
   }
   else if(((int)key)==77)
   {
    scx=1.1;
    scy=1.0;
    cleardevice();
    sca(sq,scx,scy,sx,sy);
   }
  }
  closegraph();
}


void scale_gen()
{
  char key;
```

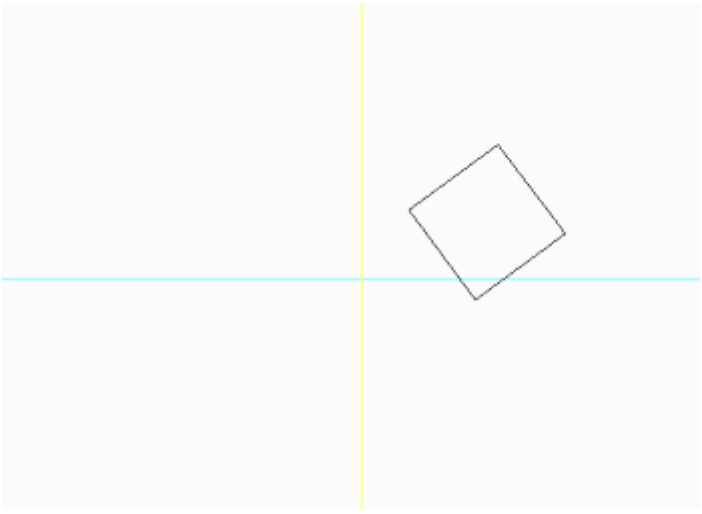
```
  int sq2[4][2];
  initia(sq);
  float scx,scy;
  while((key=getch())!='e')
  {
   if(((int)key)==72)
   {
    scy=1.1;
    scx=1.0;
    cleardevice();
    sca(sq,scx,scy,0,0);
   }
   else if(((int)key)==80)
   {
    scy=1.0/1.1;
    scx=1.0;
    cleardevice();
    sca(sq,scx,scy,0,0);
   }
   else if(((int)key)==75)
   {
    scx=1.0/1.1;
    scy=1.0;
    cleardevice();
    sca(sq,scx,scy,0,0);
   }
   else if(((int)key)==77)
   {
    scx=1.1;
    scy=1.0;
    cleardevice();
    sca(sq,scx,scy,0,0);
   }
  }
  closegraph();
}

void main()
{
  clrscr();
  int ch;
  do
  {
   cout<<"\t 2D Transformation";
   cout<<"\n\n 1. Rotation about origin \n 2. Rotation about any point \n 3. Translation\n 4. Scaling with no
fixed point\n 5. Scaling with fixed point\n 6. Exit";
   cout<<"\n\n Enter Choice : ";
   cin>>ch;
   switch(ch)
   {
    case 1 : rot_gen();
             break;
    case 2 : rot_pt();
             break;
    case 3 : translation();
             break;
```

```
    case 4 : scale_gen();
                break;
    case 5 : scale_fix();
                break;
    case 6 : break;
    default : cout<<"invalid choice ";
   }
  }while(ch!=6);
}
```

OUTPUT

```
        2D Transformation

1. Rotation about origin
2. Rotation about any point
3. Translation
4. Scaling with no fixed point
5. Scaling with fixed point
6. Exit

Enter Choice : 1
```





3D TRANSFORMATIONS

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

void put_line(int x1, int y1, int x2, int y2)
{
 line(x1+320, 200-y1,x2+320, 200-y2);
}

void disp(int pyr[8][3])
{
 for(int i=0;i<3;i++)
 {
  put_line(pyr[i][0],pyr[i][1],pyr[i+1][0],pyr[i+1][1]);
 }
 put_line(pyr[3][0],pyr[3][1],pyr[0][0],pyr[0][1]);

 for(i=4;i<7;i++)
 {
  put_line(pyr[i][0],pyr[i][1],pyr[i+1][0],pyr[i+1][1]);
 }
 put_line(pyr[7][0],pyr[7][1],pyr[4][0],pyr[4][1]);

 for(i=0;i<4;i++)
 {
  put_line(pyr[i][0],pyr[i][1],pyr[i+4][0],pyr[i+4][1]);
 }

}

void rotate( int pyr[8][3], int pyr2[8][3], float d, char ax)
{

 if(ax=='x')
 for(int i=0;i<8;i++)
 {
  pyr2[i][0]=  pyr[i][0];
  pyr2[i][1]=  pyr[i][1]*cos(d) - pyr[i][2]*sin(d);
  pyr2[i][2]=  pyr[i][1]*sin(d) + pyr[i][2]*cos(d);
 }

 else if(ax=='y')
 for(int i=0;i<8;i++)
 {
  pyr2[i][0]=  pyr[i][0]*cos(d) - pyr[i][2]*sin(d);
  pyr2[i][1]=  pyr[i][1];
  pyr2[i][2]=  pyr[i][0]*sin(d) + pyr[i][2]*cos(d);
 }

 else if(ax=='z')
 for(int i=0;i<8;i++)
 {
```

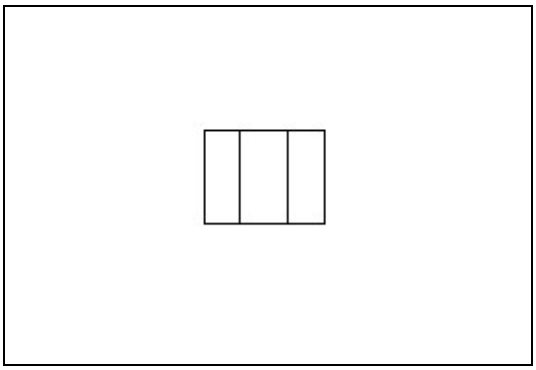
```
  pyr2[i][0]=  pyr[i][0]*cos(d) - pyr[i][1]*sin(d) ;
  pyr2[i][1]=  pyr[i][0]*sin(d) + pyr[i][1]*cos(d);
  pyr2[i][2]=  pyr[i][2];
 }
 disp(pyr2);
}

void rota(int pyr[8][3], int pyr2[8][3])
{
 cleardevice();
 cout<<" Press e to return to menu ";
 disp(pyr);
 float d=0;
 char axis='x',ax='x';
 char key;
 while((key=getch())!='e')
 {
  if(key=='x')
  {
   axis='x';
  }
  else if(key=='y')
  {
   axis='y';
  }
  else if(key=='z')
  {
   axis='z';
  }

  if((int)key==75 && ax==axis)
  {
   d+=.1;
   cleardevice();
   rotate(pyr,pyr2,d,axis);
   ax=axis;
  }
  else if((int)key==75 && ax!=axis)
  {
   d=.1;
   cleardevice();
   for(int i=0;i<8;i++)
   {
         pyr[i][0]=pyr2[i][0];
         pyr[i][1]=pyr2[i][1];
         pyr[i][2]=pyr2[i][2];
   }
   rotate(pyr,pyr2,d,axis);
   ax=axis;
  }

  else if((int)key==77 && ax==axis)
  {
   d-=.1;
   cleardevice();
   rotate(pyr,pyr2,d,axis);
```

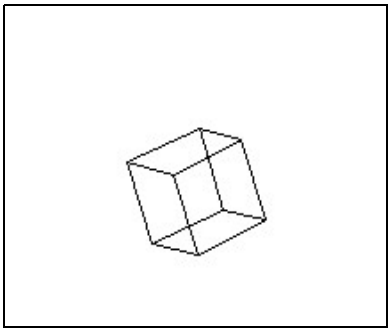
```
    ax=axis;
   }
   else if((int)key==77 && ax!=axis)
   {
    d=-0.1;
    cleardevice();
    for(int i=0;i<8;i++)
    {
          pyr[i][0]=pyr2[i][0];
          pyr[i][1]=pyr2[i][1];
          pyr[i][2]=pyr2[i][2];
    }
    rotate(pyr,pyr2,d,axis);
    ax=axis;
   }
  }
  for(int i=0;i<8;i++)
  {
   pyr[i][0]=pyr2[i][0];
   pyr[i][1]=pyr2[i][1];
   pyr[i][2]=pyr2[i][2];
  }
}

void translation(int pyr[8][3])
{
 cleardevice();
 disp(pyr);
 cout<<"Enter x translation : ";
 int tx;
 cin>>tx;
 cout<<"Enter y translation : ";
 int ty;
 cin>>ty;
 cout<<"Enter z translation : ";
 int tz;
 cin>>tz;

 cleardevice();
 for(int i=0;i<8;i++)
 {
  pyr[i][0]+=tx;
  pyr[i][1]+=ty;
  pyr[i][2]+=tz;
 }
 disp(pyr);
 getch();
 cleardevice();

}

void scale(int pyr[8][3])
{
 cleardevice();
 disp(pyr);
 cout<<"Enter x scaling factor : ";
```

```
 float tx;
 cin>>tx;
 cout<<"Enter y scaling factor : ";
 float ty;
 cin>>ty;
 cout<<"Enter z scaling factor : ";
 float tz;
 cin>>tz;

 cleardevice();
 for(int i=0;i<8;i++)
 {
  pyr[i][0]*=tx;
  pyr[i][1]*=ty;
  pyr[i][2]*=tz;
 }
 disp(pyr);
 getch();
 cleardevice();
}

void main()
{
 int pyr[8][3] = {{0,0,0},{50,0,0},{50,50,0},{0,50,0},{0,0,50},{50,0,50},{50,50,50},{0,50,50}};
 int pyr2[8][3]= {{0,0,0},{50,0,0},{50,50,0},{0,50,0},{0,0,50},{50,0,50},{50,50,50},{0,50,50}};
 char key;
 const int rot=1;
 int rotside;
 int gd=DETECT, gm;
 initgraph(&gd,&gm,"c:\\tc\\bgi");
 int ch;
 do
  {
   cleardevice();
   gotoxy(2,2);
   cout<<"\t 3D Transformation";
   cout<<"\n\n 1. Rotation \n 2. Translation\n 3. Scaling \n 4. Exit";
   cout<<"\n\n Enter Choice : ";
   cin>>ch;
   switch(ch)
   {
    case 1 : rota(pyr,pyr2);
             break;
    case 2 : translation(pyr);
             break;
    case 3 : scale(pyr);
             break;
    case 4 : break;
    default : cout<<"invalid choice ";
   }
  }while(ch!=4);
}
```

OUTPUT

```
        3D Transformation

1. Rotation
2. Translation
3. Scaling
4. Exit

Enter Choice : 1
```





ANTIALIASING – GUPTA SPROULL'S APPROACH

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

int main(void)
{
 int gdriver = DETECT, gmode, errorcode;
 float x1,x2,y1,y2,x,y,dx,dy,d,t;
 double id,di;
 void intensepix(float ,float ,double);
 initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
 printf("enter the end points of line");
 cin>>x1>>y1>>x2>>y2;
 line(20+x1,240-y1,20+x2,240-y2);
 dx=x2-x1,dy=y2-y1;
 id=1/(2*pow((dx*dx+dy*dy),0.5));
 di=2*dx*id;
 d=2*dy-dx;
 x=x1;
 y=y1;

 getch();

 intensepix(x,y,0);
 intensepix(x,y+1,di);
 intensepix(x,y-1,di);
 while(x<x2)
 {
  if(d<0)
  {
   t=d+dx;
   d+=2*dy;
   x++;
  }
  else
  {
   t=d-dx;
   d+=2*(dy-dx);
   x++,y++;
  }
  intensepix(x,y,t*id);
  intensepix(x,y+1,di-t*id);
  intensepix(x,y-1,di+t*id);
 }
 getch();
}

void intensepix(float x,float y,double d)
{
 int in;
```

```
  if(d==0.0)
    in=15;
  if((d<.4)&&(d>0))
    in=7;
  if((d<=1.0)&&(d>=.4))
    in=8;
  if(d>1)
    in=0;
  putpixel(x+320,240-y,in);
}
```

OUTPUT

```
enter the end points of lineZ0 Z0 100 100
```

aliased line

anti-aliased line

HIDDEN SURFACE ELIMINATION – Z BUFFER

```
#include<stdlib.h>
#include<math.h>
#include<conio.h>
#include<stdio.h>
#include<graphics.h>
#define PI 22/7

struct tedge
{
 int yupper ;
 float xintersect,dxperscan ;
 struct tedge *next ;
};

typedef struct tedge edge;

struct dcpt1
{
 int x ;
 int y ;
 int z;
};

typedef struct dcpt1 dcpt;
dcpt **polys; /*holds the vertices of the polygons*/
float **planes; /*holds the coeffs. of the plane equations*/
int poly_no;   /*polygon under consideration*/
int xmin,ymin,xmax,ymax ;
int zbuffer[100][100];
int screen[100][100];

void insertedge(edge *list,edge *edge1)
{
 edge *p, *q=list ;
 p =  q->next ;
 while(p!=NULL)
 {
   if (edge1->xintersect < p->xintersect)
     p = NULL ;
   else
   {
     q = p ;
     p = p->next ;
   }
 }
 edge1->next = q->next ;
 q->next = edge1;
}

int ynext(int k,int cnt,dcpt *pts)
{
 int j ;
```

```
 if ((k+1)>(cnt-1))
  j = 0 ;
 else
  j = k+1 ;

 while(pts[k].y == pts[j].y)
  if ((j+1)>(cnt-1))
   j = 0 ;
  else
   j++ ;
 return (pts[j].y) ;
}

void makeedgerec(dcpt lower,dcpt upper,int ycomp,edge *edge1,edge *edges[])
{
 edge1->dxperscan = (float)(upper.x-lower.x)/(upper.y-lower.y) ;
 edge1->xintersect = lower.x ;
 if (upper.y<ycomp)
  edge1->yupper = upper.y-1 ;
 else
  edge1->yupper = upper.y ;
 insertedge(edges[lower.y],edge1) ;
}

void buildedgelist(int cnt,dcpt *pts,edge *edges[])
{
 edge *edge1 ;
 dcpt v1, v2 ;
 int i, yprev=pts[cnt-2].y ;

 v1.x = pts[cnt-1].x ;
 v1.y = pts[cnt-1].y ;
 for(i=0;i<cnt;i++)
 {
  v2 = pts[i] ;
  if (v1.y != v2.y)
  {
   edge1 = (edge*)malloc(sizeof(edge)) ;
   if (v1.y<v2.y)
        makeedgerec(v1,v2,ynext(i,cnt,pts),edge1,edges) ;
   else
        makeedgerec(v2,v1,yprev,edge1,edges) ;
  }
  yprev = v1.y ;
  v1 = v2 ;
 }
}

void buildactivelist(int scan,edge *active,edge *edges[])
{
 edge *p,*q ;
 p = edges[scan]->next ;
 while(p)
 {
  q = p->next ;
  insertedge(active,p) ;
```

```
    p = q ;
  }
}

void checkZbuff(int x, int y, int color)
{
  int z;
  if(planes[poly_no][2]!=0)
     z=(-1.0)*(planes[poly_no][0]*x*1.0+planes[poly_no][1]*y*1.0+planes[poly_no][3])/planes[poly_no]
[2];
  else
     z=32767;
  if(z>=zbuffer[x][y])
  {
     screen[x][y]=color;
     zbuffer[x][y]=z;
  }
}


void fillscan(int scan,edge *active, int color)
{
  edge *p1,*p2 ;
  int i ;
  p1 = active->next ;
  while(p1)
  {
    p2 = p1->next ;
    for(i=p1->xintersect;i<=(p2->xintersect);i++)
    {  /*putpixel((int)i,scan,color) ;*/
       checkZbuff(i,scan,color);
    }
    p1 = p2->next ;
  }
}

void deleteafter(edge *q)
{
  edge *p = q->next ;
  q->next = p->next ;
  free(p) ;
}

void updateactivelist(int scan, edge *active)
{
  edge *q=active, *p=active->next ;
  while(p)
    if (scan >= p->yupper)
    {
      p = p->next ;
      deleteafter(q) ;
    }
    else
    {
      p->xintersect = p->xintersect + p->dxperscan ;
      q = p ;
```

```
      p = p->next ;
    }
}

void resortactivelist(edge *active)
{
 edge *q,*p=active->next ;
 active->next = NULL ;
 while(p)
 {
  q = p->next ;
  insertedge(active,p) ;
  p = q ;
 }
}

void scanfill(int cnt,dcpt *pts, int color)
{
 edge *edges[480],*active ;
 int i,scan ;
 for(i=ymax;i<ymin;i++)
 {
  edges[i] =(edge *)malloc(sizeof(edge)) ;
  edges[i]->next = NULL ;
 }
 buildedgelist(cnt,pts,edges) ;
 active = (edge *)malloc(sizeof(edge)) ;
 active->next = NULL ;

 for(scan=ymax;scan<ymin;scan++)
 {
  buildactivelist(scan,active,edges) ;
  if (active->next)
  {
   fillscan(scan,active,color) ;
   updateactivelist(scan,active) ;
   resortactivelist(active) ;
  }
 }
}

void scanZ(dcpt *ver, int n, int color)
{
 int i;
 ymax = ver[0].y ;
 ymin = ver[0].y ;
 xmin = ver[0].x ;
 xmax = ver[0].x ;
 for(i=1;i<n;i++)
 {
  if (ymin<ver[i].y)
    ymin = ver[i].y ;
  if (ymax>ver[i].y)
    ymax = ver[i].y ;
 }
 scanfill(n,ver,color) ;
```

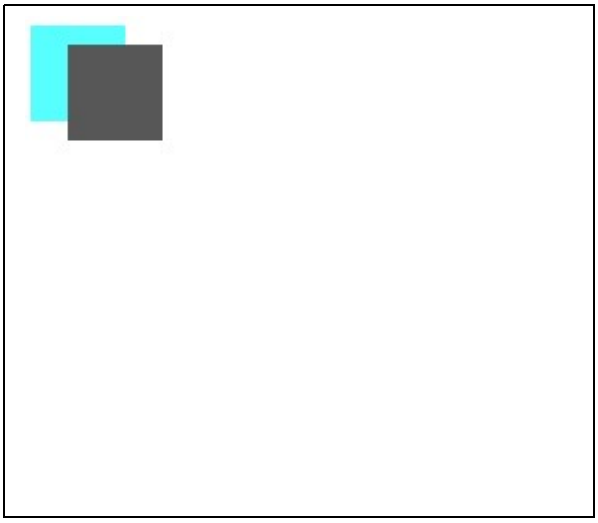
```
}

void main()
{
  int gdriver = DETECT, gmode, errorcode;
  int *color;
  char ch;
  int flag;
  int no_poly;
  int i,j;
  int k,l,m;
  int *n;
  float angle=0;
  float **temp;
  float matrix[4][4];
  float rad_angle;
  float x1,y1,z1,x2,y2,z2,x3,y3,z3;
  clrscr();
 matrix[0][3]=0;
 matrix[1][3]=0;
 matrix[2][3]=0;
 matrix[3][3]=1;
 matrix[3][0]=0;
 matrix[3][1]=0;
 matrix[3][2]=0;
  printf("Enter number of polygons:");
  scanf("%d",&no_poly);
  temp=(float**)malloc(4*sizeof(float));
  planes=(float**)malloc(no_poly*sizeof(float));
  polys=(dcpt**)malloc(no_poly*sizeof(dcpt));
  color=(int*)malloc(no_poly*sizeof(int));
  n=(int*)malloc(no_poly*sizeof(int));
  for(i=0;i<100;++i)
    for(j=0;j<100;++j)
    {
      zbuffer[i][j]=-32768;
      screen[i][j]=BLACK;
    }
  for(i=0;i<no_poly;++i)
  {
    printf("\nEnter the number of sides of polygon number %d: ",i+1) ;
    scanf("%d",&n[i]) ;
    printf("\nColor of this polygon?:");
    scanf("%d",&color[i]);
    if (n[i]<=2)
    {
      printf("\n Invalid Entry " ) ;
      printf("\n Press any key to Exit " );
      getch() ;
      exit(1) ;
    }
    polys[i]=(dcpt*)malloc(sizeof(dcpt)*n[i]);
    planes[i]=(float*)malloc(4*sizeof(float));
    for(j=0;j<n[i];++j)
    {
      printf("Enter x%d,y%d,z%d:",j+1,j+1,j+1);
```

```
    scanf("%d %d %d",&(polys[i][j].x),&(polys[i][j].y),&(polys[i][j].z));
  }
}
clrscr();
for(i=0;i<no_poly;++i)
  for(j=0;j<4;++j)
    temp[j]=(float*)malloc(n[i]*sizeof(float));
initgraph(&gdriver,&gmode,"c:\\tc\\bgi");
do
{
for(i=0;i<no_poly;++i)
{
  x1=polys[i][0].x;y1=polys[i][0].y;z1=polys[i][0].z;
  x2=polys[i][1].x;y2=polys[i][1].y;z2=polys[i][1].z;
  x3=polys[i][2].x;y3=polys[i][2].y;z3=polys[i][2].z;
  /*A*/
  planes[i][0]=y1*(z2-z3)+y2*(z3-z1)+y3*(z1-z2);
  /*B*/
  planes[i][1]=z1*(x2-x3)+z2*(x3-x1)+z3*(x1-x2);
  /*C*/
  planes[i][2]=x1*(y2-y3)+x2*(y3-y1)+x3*(y1-y2);
  /*D*/
  planes[i][3]=(-1)*x1*(y2*z3-y3*z2)-x2*(y3*z1-y1*z3)-x3*(y1*z2-y2*z1);
}
for(poly_no=0;poly_no<no_poly;++poly_no)
   scanZ(polys[poly_no],n[poly_no],color[poly_no]);
for(i=0;i<100;++i)
  for(j=0;j<100;++j)
    putpixel(i,j,screen[i][j]);
ch=getch();
cleardevice();
switch(ch)
{
   case 75:angle=1;break;
   case 77:angle=-1;break;
   default:flag=1;break;
}
if(flag)
{
  flag=0;
  continue;
}
rad_angle=angle*PI/180.0;
matrix[0][0]=cos(rad_angle);
matrix[0][2]=sin(rad_angle);
matrix[2][0]=(-1)*sin(rad_angle);
matrix[2][2]=cos(rad_angle);
matrix[0][1]=0;matrix[1][0]=0;matrix[1][1]=1;matrix[1][2]=0;matrix[2][1]=0;
for(i=0;i<no_poly;++i)
{
  for(l=0;l<n[i];++l)
  {
    temp[0][l]=polys[i][l].x;
    temp[1][l]=polys[i][l].y;
    temp[2][l]=polys[i][l].z;
    temp[3][l]=1;
```

```
      polys[i][l].x=0;
      polys[i][l].y=0;
      polys[i][l].z=0;
     }
     for(l=0;l<n[i];++l)
      for(m=0;m<4;m++)
      {
            (polys[i][l].x)+=(matrix[0][m]*temp[m][l]);
            (polys[i][l].y)+=(matrix[1][m]*temp[m][l]);
            (polys[i][l].z)+=(matrix[2][m]*temp[m][l]);
      }
   }
   }while(ch!='e');
   getch();
   cleardevice();
   closegraph();
}
```

OUTPUT

```
Enter number of polygons:2

Enter the number of sides of polygon number 1: 4

Color of this polygon?:4
Enter x1,y1,z1:10 10 0
Enter x2,y2,z2:60 10 0
Enter x3,y3,z3:60 60 0
Enter x4,y4,z4:10 60 0

Enter the number of sides of polygon number 2: 4

Color of this polygon?:7
Enter x1,y1,z1:30 20 10
Enter x2,y2,z2:80 20 10
Enter x3,y3,z3:80 70 10
Enter x4,y4,z4:30 70 10
```



BACK FACE DETECTION

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

float transform[4][4];
enum {x,y,z};

struct point
{
   float x,y,z;
};

point cube[6][4] =
{{{-10,10,-10},{-10,10,10},{10,10,10},{10,10,-10}},
 {{10,10,10},{10,10,-10},{10,-10,-10},{10,-10,10}},
 {{-10,10,10},{-10,10,-10},{-10,-10,-10},{-10,-10,10}},
 {{-10,10,10},{10,10,10},{10,-10,10},{-10,-10,10}},
 {{-10,10,-10},{10,10,-10},{10,-10,-10},{-10,-10,-10}},
 {{-10,-10,10},{10,-10,10},{10,-10,-10},{-10,-10,-10}}};

void initialise()
{
        int i,j;
        for(i=0;i<4;i++)
            for(j=0;j<4;j++)
                if(i==j)
                  transform[i][j]=1;
                else
                  transform[i][j] = 0;
}

void updatecube(float ttransform[4][4])
{
        float coor[4][1],temp[4][1];
        int i,j,k,l;
        for(i=0;i<6;i++)
           for(j=0;j<4;j++)
           {
                coor[0][0]=cube[i][j].x;
                coor[1][0]=cube[i][j].y;
                coor[2][0]=cube[i][j].z;
                coor[3][0]=1;
                for(k=0;k<4;k++)
                    temp[k][0]=0;
                for(l=0;l<4;l++)
                     for(k=0;k<4;k++)
                          temp[l][0]+=ttransform[l][k]*coor[k][0];
                cube[i][j].x=temp[0][0];
                cube[i][j].y=temp[1][0];
                cube[i][j].z=temp[2][0];
                }
}
```

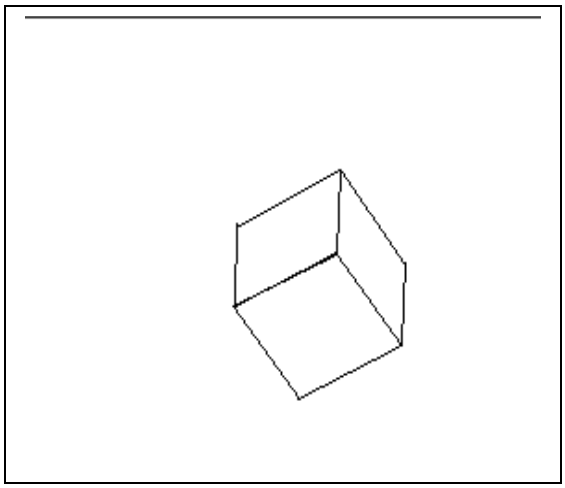
```
void updatetransform(float ttransform[4][4])
{
        int i,j,k;
        float temp[4][4];
        for(i=0;i<4;i++)
             for(j=0;j<4;j++)
                  temp[i][j]=0;
        for(i=0;i<4;i++)
             for(j=0;j<4;j++)
                  for(k=0;k<4;k++)
                      temp[i][j]+=ttransform[i][k]*transform[k][j];
        for(i=0;i<4;i++)
            for(j=0;j<4;j++)
                transform[i][j]=temp[i][j];
}

void rotate(float angle,int raxis)
{
        float fsin,fcos,ttransform[4][4];
        int i,j;
        for(i=0;i<4;i++)
            for(j=0;j<4;j++)
                  if(i==j)
                    ttransform[i][j]=1;
                  else ttransform[i][j] = 0;
        fsin=sin(angle);
        fcos=cos(angle);
        if(raxis==0)
        {
                  ttransform[1][1] = fcos;
                  ttransform[1][2] = -1*fsin;
                  ttransform[2][1] = fsin;
                  ttransform[2][2] = fcos;
        }
        else if(raxis==1)
            {
                  ttransform[0][0] = fcos;
                  ttransform[0][2] = fsin;
                  ttransform[2][0] = -1*fsin;
                  ttransform[2][2] = fcos;
            }
            else
            {
                  ttransform[0][0] = fcos;
                  ttransform[0][1] = -1*fsin;
                  ttransform[1][0] = fsin;
                  ttransform[1][1] = fcos;
            }
        updatetransform(ttransform);
        updatecube(ttransform);
}

void translate(float Delx,float Dely,float Delz)
{
        float ttransform[4][4];
```

```
        int i,j;
        for(i=0;i<4;i++)
          for(j=0;j<4;j++)
            if(i==j)
                ttransform[i][j]=1;
            else
                ttransform[i][j]=0;
        ttransform[0][3]=Delx;
        ttransform[1][3]=Dely;
        ttransform[2][3]=Delz;
        updatetransform(ttransform);
}


void scale(float Sx,float Sy,float Sz)
{
        float ttransform[4][4];
        int i,j;
        for(i=0;i<4;i++)
            for(j=0;j<4;j++)
                if(i==j)
                  ttransform[i][j]=1;
                else
                  ttransform[i][j]=0;
        ttransform[0][0]=Sx;
        ttransform[1][1]=Sy;
        ttransform[2][2]=Sz;
        updatetransform(ttransform);
        updatecube(ttransform);
}

float dotproduct(struct point a,struct point b)
{
        int result;
        result=a.x*b.x+a.y*b.y+a.z*b.z;
        return(result);
}

struct point crossproduct(struct point a,struct point b)
{
        struct point result;
        result.x=b.z*a.y-b.y*a.z;
        result.y=a.z*b.x-a.x*b.z;
        result.z=a.x*b.y-b.x*a.y;
        return(result);
}

void o_project()
{
        int i,j;
        float ttransform[4][4];
        for(i=0;i<4;i++)
            for(j=0;j<4;j++)
                if(i==j)
                  ttransform[i][j]=1;
                else
```

```
                ttransform[i][j]=0;
        ttransform[2][2]=0;
        updatetransform(ttransform);
}

void draw(int surface)
{
        int midx,midy,i,j,k,prevx,prevy;
        float coor[4][1],temp[4][1],temp1[3];
        int flag=0;
        midx=getmaxx()/2;
        midy=getmaxy()/2;
        for(i=0;i<4;i++)
        {
                coor[0][0]=cube[surface][i].x;
                coor[1][0]=cube[surface][i].y;
                coor[2][0]=cube[surface][i].z;
                coor[3][0]=1;
                for(j=0;j<4;j++)
                    temp[j][0]=0;
                for(j=0;j<4;j++)
                  for(k=0;k<4;k++)
                     temp[j][0]+=transform[j][k]*coor[k][0];
                if(flag!=0)
           line(midx+prevx,midy-prevy,midx+coor[0][0],midy-coor[1][0]);
                else
                {
                        flag = 1;
                        temp1[0]=coor[0][0];
                        temp1[1]=coor[1][0];
                        temp1[2]=coor[2][0];
                }
                prevx=coor[0][0];
                prevy=coor[1][0];
        }
        line(midx+prevx,midy-prevy,midx+temp1[0],midy-temp1[1]);
}

void detect_surface()
{
        struct point n1,n2,n3;
        int i,j;
        struct point v={0,0,-1};
        for(i=0;i<6;i++)
        {
           n2.x=cube[i][1].x-cube[i][0].x;
           n2.y=cube[i][1].y-cube[i][0].y;
           n2.z=cube[i][1].z-cube[i][0].z;
           n3.x=cube[i][3].x-cube[i][0].x;
           n3.y=cube[i][3].y-cube[i][0].y;
           n3.z=cube[i][3].z-cube[i][0].z;
           n1=crossproduct(n2,n3);
           n2=cube[i][0];
           if(dotproduct(n1,n2)<0)
           {
                n1.x=-1*n1.x;
```

```
                    n1.y=-1*n1.y;
                    n1.z=-1*n1.z;
                }
                if(dotproduct(n1,v)<0)
                draw(i);
            }
}

void main()
{
        float angle;
        int i,j;
        char ch;
        int gd = DETECT, gmode;
        initgraph( &gd, &gmode, "c:\\tc\\bgi");

        initialise();
        angle=0.05;
        ch=0;
        while(ch!=27)
        {
          cleardevice();
          initialise();
          if(ch=='M')
             rotate(angle,y);
          if(ch=='K')
             rotate(-1*angle,y);
          if(ch=='P')
             rotate(angle,x);
          if(ch=='H')
             rotate(-1*angle,x);
          if(ch=='+')
             scale(1.005,1.005,1.005);
          if(ch=='-')
             scale(0.995,0.995,0.995);
          initialise();
          detect_surface();
          o_project();
          fflush(stdin);
          ch=getch();
        }
        getch();
        closegraph();
}
```

OUTPUT



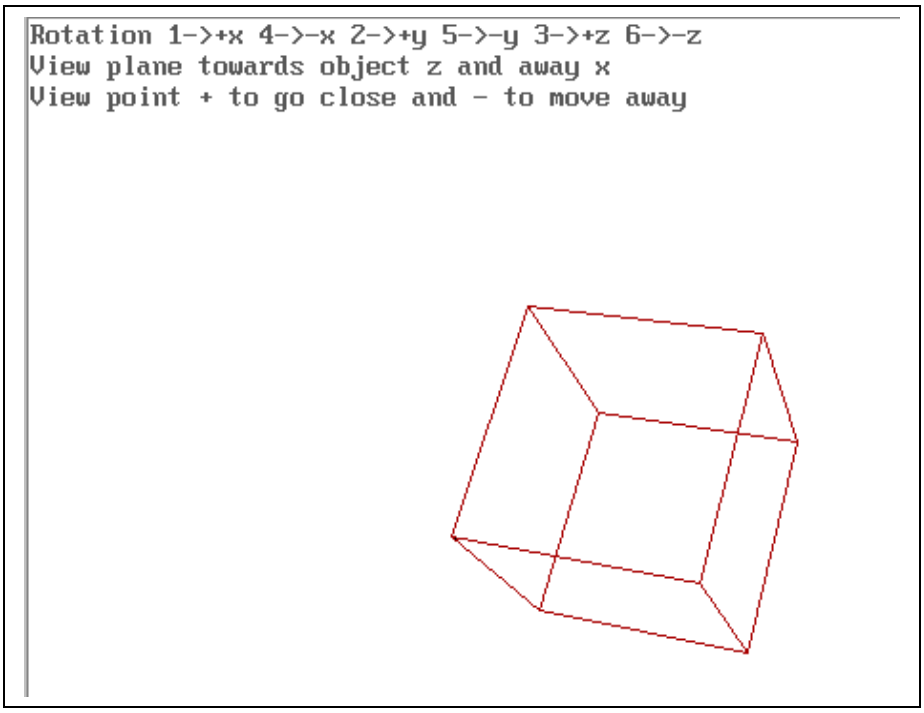
PROGRAM TO VIEW A 3D FIGURE IN PERSPECTIVE

```
#include<graphics.h>
#include<math.h>
#define D 70
#define PI 3.14

float cor[8][3]={  -D,-D,-D,
                   -D,D,-D,
                   D,D,-D,
                   D,-D,-D,
                   -D,D,D,
                   D,D,D,
                   D,-D,D,
                   -D,-D,D
                 } ;
int ox,oy;
long zprp=-500,zvp=-D;
float zmax=0;
void main()
{
   int gd=DETECT,gm;
   void rotx(float);
   void roty(float);
   void rotz(float);
   void draw();
   char key=0,i;
   initgraph(&gd,&gm,"c:\\tc\\bgi");
   ox=getmaxx()/2;
   oy=getmaxy()/2;
   outtextxy(10,10,"Rotation 1->+x 4->-x 2->+y 5->-y 3->+z 6->-z");
   outtextxy(10,20,"View plane towards object z and away x");
   outtextxy(10,30,"View point + to go close and - to move away");
   do
   {
         setcolor(0);
         draw();
         if(key=='+'&&zvp-zprp>5)
           zprp+=4;
         if(key=='-'&&zprp>-1000)
           zprp-=4;
         if(key=='z'&&zvp<5*D)
           zvp+=4;
         if(key=='x'&&zvp-zprp>5)
           zvp-=4;
         if(key=='1')
           rotx(5.0);
         if(key=='4')
           rotx(-5.0);
         if(key=='5')
```

```
            roty(-5.0);
          if(key=='2')
            roty(5.0);
          if(key=='3')
            rotz(5.0);
          if(key=='6')
            rotz(-5.0);
          zmax=-3*D;
          for(i=0;i<8;i++)
            if(cor[i][2]>zmax)
              zmax=cor[i][2];
          setcolor(9);
          draw();
          key=getch();
    } while(key!=27);
}

void draw()
{
    void dline(char,char);
    dline(0,1);
    dline(0,3);
    dline(0,7);
    dline(5,6);
    dline(5,2);
    dline(5,4);
    dline(2,3);
    dline(1,4);
    dline(4,7);
    dline(7,6);
    dline(3,6);
    dline(1,2);
}

void dline(char p1,char p2)
{
    int x1,x2,y1,y2,d;
    d=zprp-zvp;
    x1=cor[p1][0]*d/(cor[p1][2]-zprp);
    x2=cor[p2][0]*d/(cor[p2][2]-zprp);
    y1=cor[p1][1]*d/(cor[p1][2]-zprp);
    y2=cor[p2][1]*d/(cor[p2][2]-zprp);
    line(x1+ox,y1+oy,x2+ox,y2+oy);
}


void rotx(float d)
{
    int i;
    float y,z;
    float cs,sn;
```

```
    cs=cos(d*PI/180);
    sn=sin(d*PI/180);
    for(i=0;i<8;i++)
    {
            y=cor[i][1]*cs-cor[i][2]*sn;
            z=cor[i][1]*sn+cor[i][2]*cs;
            cor[i][1]=y;
            cor[i][2]=z;
    }
}

void roty(float d)
{
    int i;
    float x,z;
    float cs,sn;
    cs=cos(d*PI/180);
    sn=sin(d*PI/180);
    for(i=0;i<8;i++)
    {
            x=cor[i][2]*sn+cor[i][0]*cs;
            z=cor[i][2]*cs-cor[i][0]*sn;
            cor[i][0]=x;
            cor[i][2]=z;
    }
}

void rotz(float d)
{
    int i;
    float x,y;
    float cs,sn;
    cs=cos(d*PI/180);
    sn=sin(d*PI/180);
    for(i=0;i<8;i++)
    {
            x=cor[i][0]*cs-cor[i][1]*sn;
            y=cor[i][0]*sn+cor[i][1]*cs;
            cor[i][0]=x;
            cor[i][1]=y;
    }
}
```

OUTPUT

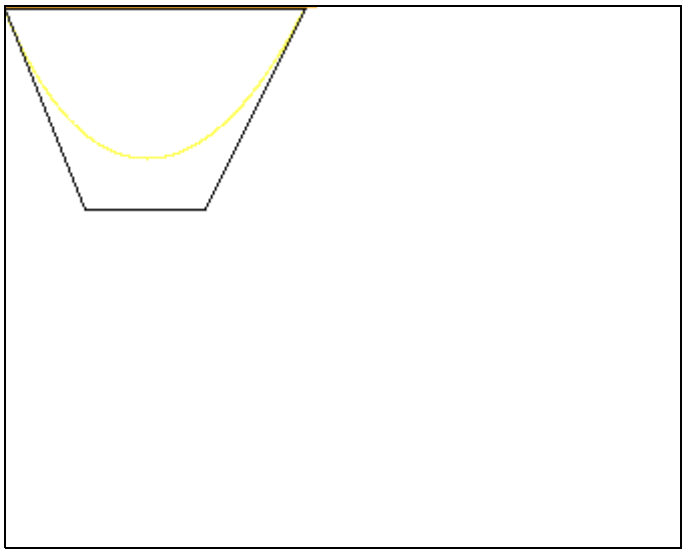

Rotation 1->+x 4->-x 2->+y 5->-y 3->+z 6->-z
View plane towards object z and away x
View point + to go close and - to move away

BEZIER CURVE

```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
void bezier();
void main()
{
 int driver,mode;
 driver=DETECT;
 initgraph(&driver,&mode,"c:\\tc\\bgi");
 bezier();
}

void bezier()
{
 float p1[3],p2[3],p3[3],p4[3],temp[3];
 cout<<"\n enter the coords of P1 \n";
 cin>>p1[0]>>p1[1]>>p1[2];
 cout<<"\n enter the coords of P2 \n";
 cin>>p2[0]>>p2[1]>>p2[2];
 cout<<"\n enter the coords of P3 \n";
 cin>>p3[0]>>p3[1]>>p3[2];
 cout<<"\n enter the coords of P4  \n";
 cin>>p4[0]>>p4[1]>>p4[2];
 temp[0]=p1[0]; temp[1]=p1[1]; temp[2]=p1[2];
 cleardevice();
 for(float t=.001;t<=1;t+=.001)
 {
  temp[0]=(1-t)*(1-t)*(1-t)*p1[0] + (3*t*(1-t)*(1-t))*p2[0] + ((3*t*t)*(1-t))*p3[0] + ((t*t*t))*p4[0];
  temp[1]=(1-t)*(1-t)*(1-t)*p1[1] + (3*t*(1-t)*(1-t))*p2[1] + ((3*t*t)*(1-t))*p3[1] + ((t*t*t))*p4[1];
  temp[2]=(1-t)*(1-t)*(1-t)*p1[2] + (3*t*(1-t)*(1-t))*p2[2] + ((3*t*t)*(1-t))*p3[2] + ((t*t*t))*p4[2];
  putpixel(temp[0],temp[1],BLUE);
 }
 setcolor(WHITE);
 line(p1[0],p1[1],p2[0],p2[1]);
 line(p2[0],p2[1],p3[0],p3[1]);
 line(p3[0],p3[1],p4[0],p4[1]);
 line(p1[0],p1[1],p4[0],p4[1]);
 getch();
}
```

OUTPUT

```
 enter the coords of P1
0 0 0

 enter the coords of P2
40 100 0

 enter the coords of P3
100 100 10

 enter the coords of P4
150 0 10
```



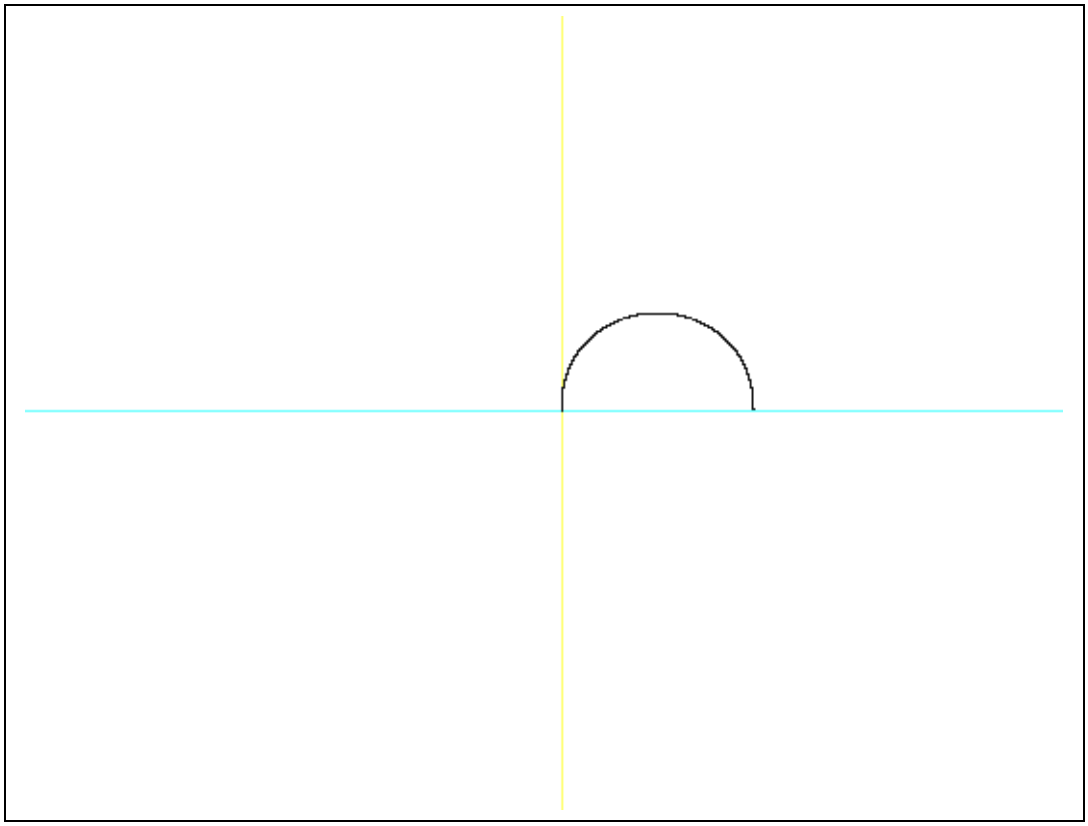
HERMITE CURVE

```
#include<graphics.h>
#include<iostream.h>
#include<process.h>
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<dos.h>

struct pt
{
 float x,y;
 float mx,my;
};
struct pt g1,g2;

void main()
{
 float outx,outy;
 float u;
 float gx[4],gy[4],tempx[4],tempy[4];

 cout<<"Enter the x-co-ord of first point: ";
 cin>>g1.x;
 cout<<"\n\nEnter the y-co-ord of first point: ";
 cin>>g1.y;
 cout<<"\n\nEnter the x-co-ord of second point: ";
 cin>>g2.x;
 cout<<"\n\nEnter the y-co-ord of the second point: ";
 cin>>g2.y;

 cout<<"Enter the x-co-ord of the tangent vector at first end point";
 cin>>g1.mx;
 cout<<"Enter the tangent vector's y co-ord at first end point: ";
 cin>>g1.my;
 cout<<"Enter the tangent vector's x-co-ord at second end point: ";
 cin>>g2.mx;
 cout<<"Enter the tangent vector's y-co-ord at second end point: ";
 cin>>g2.my;

 int gdriver = DETECT, gmode, errorcode;

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

 gx[0]=g1.x;
 gx[1]=g2.x;
 gx[2]=g1.mx;
 gx[3]=g2.mx;

 gy[0]=g1.y;
 gy[1]=g2.y;
 gy[2]=g1.my;
 gy[3]=g2.my;
```

```
  tempx[0] = 2*(gx[0]-gx[1]) + gx[2] + gx[3];
  tempx[1] = -3*(gx[0]-gx[1]) - 2*gx[2] - gx[3];
  tempx[2] = gx[2];
  tempx[3] = gx[0];

  tempy[0] = 2*(gy[0]-gy[1]) + gy[2] + gy[3];
  tempy[1] = -3*(gy[0]-gy[1]) - 2*gy[2] - gy[3];
  tempy[2] = gy[2];
  tempy[3] = gy[0];

  setcolor(RED);
  line(0,240,640,240);
  setcolor(BLUE);
  line(320,0,320,480);
  setcolor(WHITE);


  for(u=0;u<=1;u+=0.0001)
  {
    outx=u*u*u*tempx[0]+u*u*tempx[1]+u*tempx[2]+tempx[3];
    outy=u*u*u*tempy[0]+u*u*tempy[1]+u*tempy[2]+tempy[3];
    putpixel(320+outx,240-outy,15);
  }

  getch();
}
```

OUTPUT

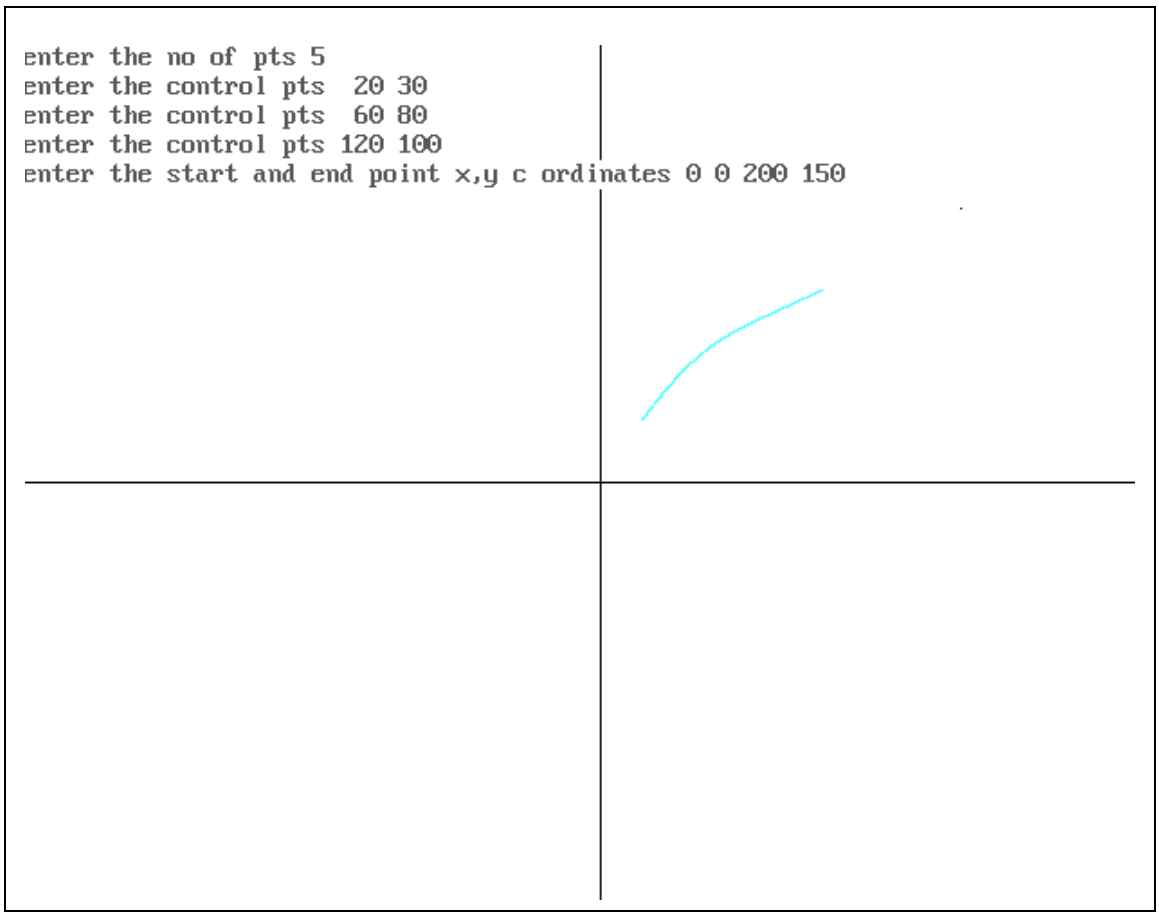
```
Enter the x-co-ord of first point: 0


Enter the y-co-ord of first point: 0


Enter the x-co-ord of second point: 100


Enter the y-co-ord of the second point: 0
Enter the x-co-ord of the tangent vector at first end point0
Enter the tangent vector's y co-ord at first end point: 200
Enter the tangent vector's x-co-ord at second end point: 0
Enter the tangent vector's y-co-ord at second end point: -200
```



B-SPLINE CURVE

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

int main(void)
{

  int gdriver = DETECT, gmode, errorcode;

  float t,i,y,x,j,k;
  int cx[20],cy[20],n;

  /* initialize graphics and local
  variables */
  initgraph(&gdriver, &gmode, "c:\\tc\\bgi");
  line(320,0,320,480);
  line(0,240,640,240);
  cout<<"enter the no of pts";
  cin>>n;
  for(i=1;i<n-1;i++)
  {
   cout<<"enter the control pts ";
   cin>>cx[i]>>cy[i];
  }
  cout<<"enter the start and end point x,y c ordinates";
  cin>>cx[0]>>cy[0]>>cx[n-1]>>cy[n-1];
  for(i=3;i<=n-1;i++)
  {
   for(t=0;t<=1;t+=.0005)
   {
          x=((pow((1-t),3))*cx[i-3]+(3*t*t*t-6*t*t+4)*cx[i-2]+(-3*t*t*t+3*t*t+3*t+1)*cx[i-
1]+t*t*t*cx[i])/6;
          y=((pow((1-t),3))*cy[i-3]+(3*t*t*t-6*t*t+4)*cy[i-2]+(-3*t*t*t+3*t*t+3*t+1)*cy[i-
1]+t*t*t*cy[i])/6;
          putpixel(320+x,240-y,RED);
          delay(1);
   }
  }
  putpixel(320+cx[0],240-cy[0],WHITE);
  putpixel(320+cx[n-1],240-cy[n-1],WHITE);
  getch();
}
```

OUTPUT


enter the no of pts 5
enter the control pts 20 30
enter the control pts 60 80
enter the control pts 120 100
enter the start and end point x,y c ordinates 0 0 200 150