Computer Graphics

NAME
ROLL NO
BRANCH AND SECTION

Program 1

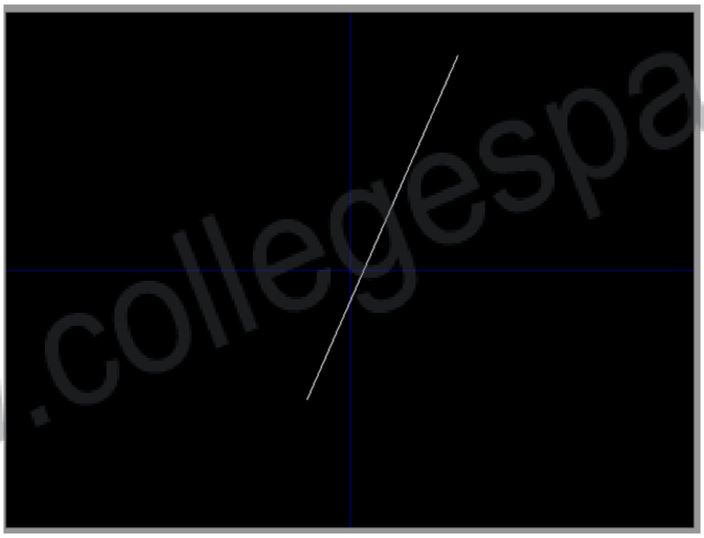
Objective:
To draw line using the DDA approach.

Code:
```
#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



Program 2

Objective:
To draw line using the Bresenham approach.

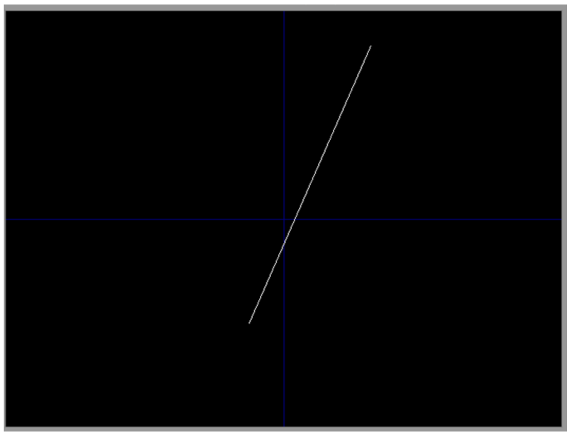
Code:

```
#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:



Program 3

Objective:

To draw line using the Mid-Point approach.

Code:

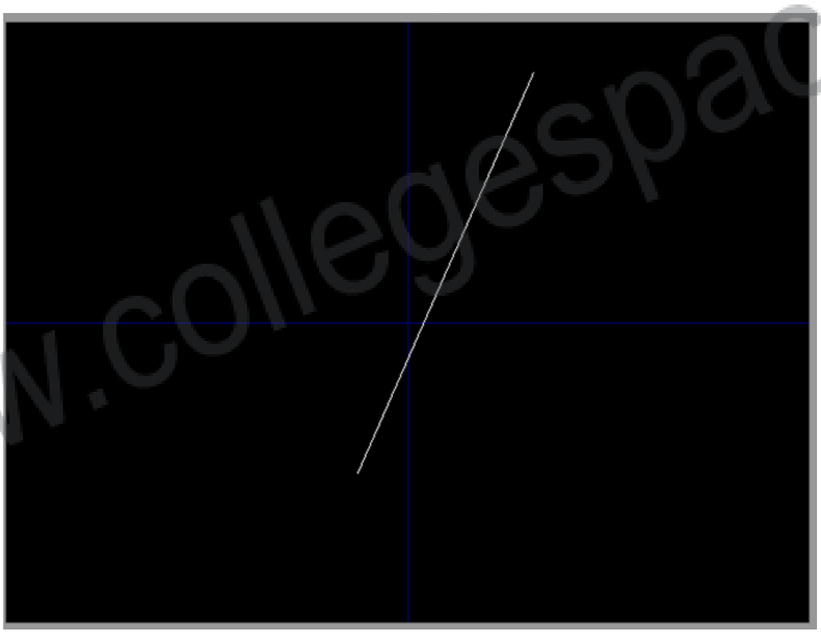
```
#include "midLine381.h"
#include <graphics.h>

void midLine(int x1, int y1, int x2, int y2, int color)
{
   if (x2 - x1 == 0)
   {
      if (y1 > y2)
      {
         int temp = y1;
         y1 = y2;
         y2 = temp;
      }
      int y = y1;
      while (y <= y2){
         putpixel(W/2+x1, H/2-y, color);
         y++;
      }
      return;
   }
   else if (y2 == y1)
   {
      if (x1 > x2){
         int temp = x1;
         x1 = x2;
         x2 = temp;
      }
      int x = x1;
      while (x <= x2){
         putpixel(W/2+x, H/2-y1, color);
         x++;
      }
      return;
}
   float m = (float)(y2-y1)/(x2-x1);
   int dx = x2-x1;
   int dy = y2-y1;
   int x = x1, y = y1;
   int d;
   putpixel(W/2+x, H/2-y, color);
```

```
if (m >= 0 && m < 1)
{
   d = 2*dy - dx;
   while (x < x2)
   {
      if (d <= 0)
         d += dy;
      else{
         d += dy - dx;
         y++;
      }
      x++;
      putpixel(W/2+x, H/2-y, color);
   }
   return;
}
else if (m > 1)
{
   d = 2*dx - dy;
   while (y < y2)
   {
      if (d <= 0)
      {
         d += dx;
      }
      else
      {
         d += dx - dy;
         x++;
      }
      y++;
      putpixel(W/2+x, H/2-y, color);
   }
   return;
}
else if (m == 1)
{
   while (y < y2)
   {
      x++;
      y++;
      putpixel(W/2+x, H/2-y, color);
   }
   return;
}
else if (m == -1)
{
   while (y > y2)
   {
```

```
            x++;
            y--;
            putpixel(W/2+x, H/2-y, color);
        }
        return;
    }
}
```

Output:

```
int main(void)
{
    initwindow(640, 480, "Varun Sharma - 557/IC/13");
    drawAxes();
midLine(-40, -120, 100, 200, WHITE);
}
```



Program 4

Objective:

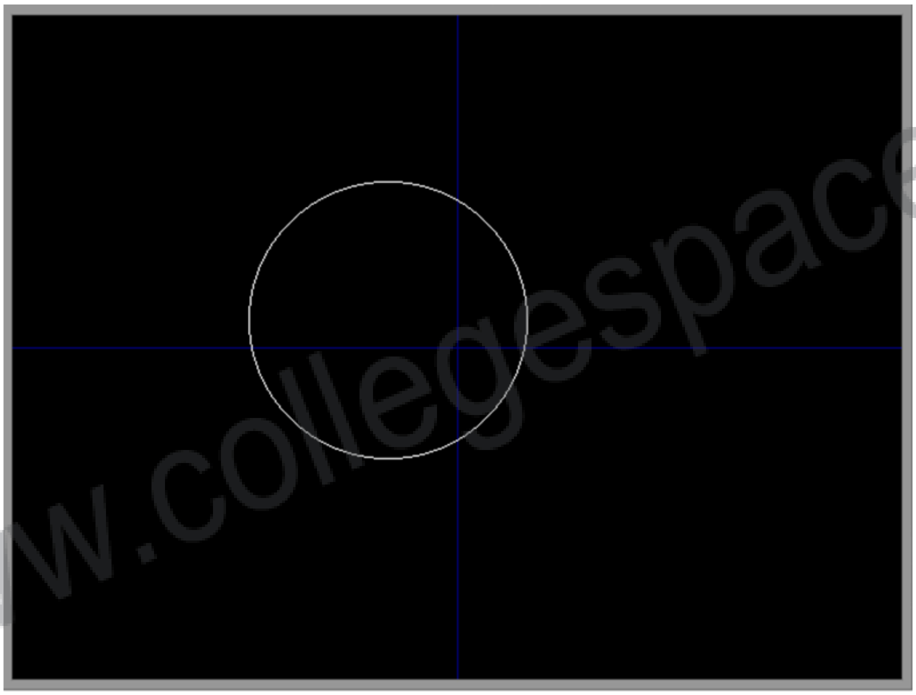
To draw a circle using the Mid-Point approach.

Code:

```
#include "midCircle381.h"
#include <graphics.h>

void plotCircle(int cx, int cy, int x, int y, int color)
{
   cx += 320;
   cy = 240 - cy;
   putpixel(cx-x,cy+y,color);
   putpixel(cx-y,cy+x,color);
   putpixel(cx+x,cy+y,color);
   putpixel(cx+y,cy+x,color);
   putpixel(cx-x,cy-y,color);
   putpixel(cx-y,cy-x,color);
   putpixel(cx+x,cy-y,color);
   putpixel(cx+y,cy-x,color);
}

void midCircle(int cx, int cy, int r, int color)
{
   int x = 0;
   int y = r;
   int d = 5/4 - r;
   plotCircle(cx, cy, x, y, color);
   while (x <= y)
   {
      if (d <= 0)
      {
         d += 2*x + 3;
      }
      else
      {
         d += 2*(x-y) + 5;
         y--;
      }
      x++;
      plotCircle(cx, cy, x, y, color);
   }
}
```

Output:

```
int main(void)
{
   initwindow(640, 480, "Varun Sharma - 557/IC/13");
   drawAxes();
midCircle(-50, 20, 100, WHITE);
}
```



Program 4

Objective:

To draw an ellipse using the Mid-Point approach.

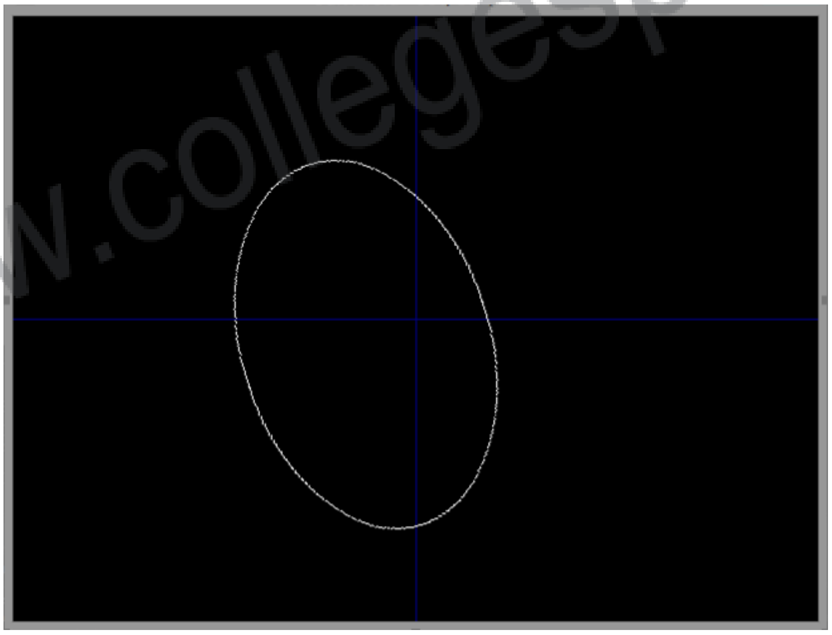
Code:

```
#include "midEllipse381.h"
#include <graphics.h>
#include <math.h>

void plotEllipse(int cx, int cy, int x, int y, double angle, int color)
{
        int x1,y1;
        x1 = (int)(x*cos(angle) - y*sin(angle));
        y1 = (int)(x*sin(angle) + y*cos(angle));
        putpixel(320 + cx + x1, 240 - (cy + y1),color);

        x1 = (int)(x*cos(angle) + y*sin(angle));
        y1 = (int)(x*sin(angle) - y*cos(angle));
        putpixel(320 + cx + x1, 240 - (cy + y1),color);

        x1 = (int)(y*sin(angle)-x*cos(angle));
        y1 = (int)(-1*x*sin(angle) - y*cos(angle));
        putpixel(320 + cx + x1, 240 - (cy + y1),color);

        x1 = (int)-1*(x*cos(angle) + y*sin(angle));
        y1 = (int)(y*cos(angle)-x*sin(angle));
        putpixel(320 + cx + x1, 240 - (cy + y1),color);

}
void midEllipse (int cx, int cy, int a, int b, float angle, int color)
{
  int x = 0, y = b;
  int d = b*b + (a*a)/4 - a*a*b;
  plotEllipse(cx, cy, x, y, angle, color);
  while ((a*a*y) >= (b*b*x))
  {
    if (d <= 0)
      d += b*b*(2*x + 3);
    else {
      d += b*b*(2*x + 3) - 2*a*a*(y-1); y--;
    }
    x++;
    plotEllipse(cx, cy, x, y, angle, color);
  }
```

```
    d = d;
    while (y >= 0)
    {
        if (d >= 0)
            d += a*a*(3 - 2*y);
        else {
            d += b*b*(2*x + 2) + a*a*(3 - 2*y); x++;
        }
        y--;
        plotEllipse(cx, cy, x, y, angle, color);
    }
}
```

Output:

```
int main(void){
    initwindow(640, 480, "Varun Sharma - 557/IC/13");
    drawAxes();
midEllipse(-40, -20, 100, 150, 0.3, WHITE);
}
```



Program 5

Objective:

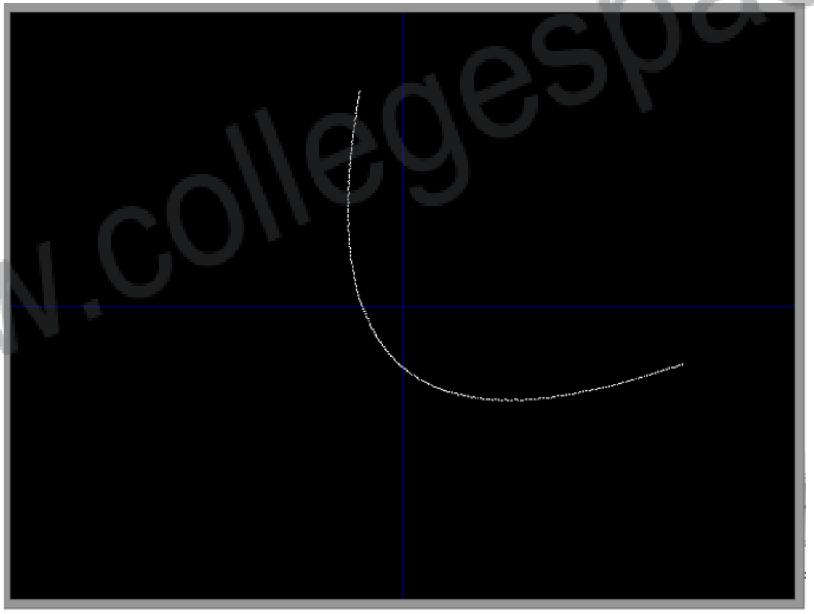
To draw parabola using Mid-Point approach.

Code:

```
#include "midParabola381.h"
#include <graphics.h>
#include <math.h>

void plotParabola(int cx, int cy, int x, int y, double angle, int color)
{
	int x1,y1;
	x1 = (int)(x*cos(angle) - y*sin(angle));
	y1 = (int)(x*sin(angle) + y*cos(angle));
	//putpixel(320 + cx + x1, 240 - (cy + y1),color); //y^2 = 4px
	putpixel(320 + (cx + y1), 240 - (cy + x1), color); //x^2 = 4py

	x1 = (int)(x*cos(angle) + y*sin(angle));
	y1 = (int)(x*sin(angle) - y*cos(angle));
	//putpixel(320 + cx + x1, 240 - (cy + y1),color); //y^2 = 4px
	putpixel(320 + (cx + y1), 240 - (cy + x1), color); //x^2 = 4py
}

void midParabola(int cx, int cy, int p, double angle, int color)
{
  int x = 0, y = 0, d = 1-p;
  plotParabola(cx, cy, x, y, angle, color);
  while (y <= p)
  {
    if (d <= 0)
      d += 2*y + 3;
    else{
      d += 2*y + 3 - 2*p;x++;
    }
    y++;
    plotParabola(cx, cy, x, y, angle, color);
  }
  d = ((float)(y+0.5))*(y+0.5) - 2*p*(x+1);
  while (x < 150)
  {
        if (d > 0)
            d += -2*p;
        else{
            d += 2*y + 2 - 2*p; y++;
        }
        x++;
        plotParabola(cx, cy, x, y, angle, color);
    }
}
```

Output:

```
int main(void){
    initwindow(640, 480, "Varun Sharma - 557/IC/13");
    drawAxes();
midParabola(0, -50, 100, 0.7, WHITE);
}
```



Program 6

Objective:

To draw ahyperbola using Bresenham's Approach

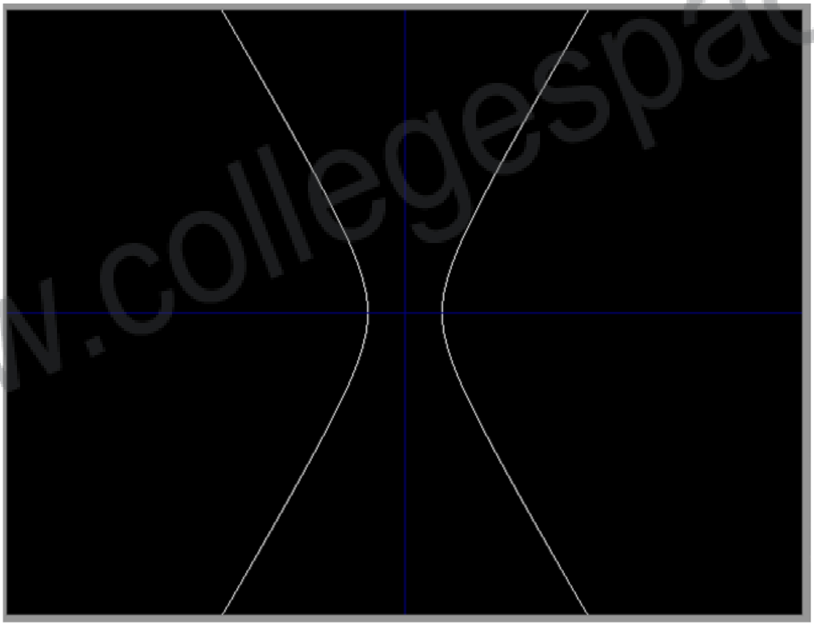
Code:

```
#include "bresHyperbola381.h"
#include <graphics.h>

void plotHyperbola(int cx, int cy, int x,int y, int color)
{
  cx += 320; cy = 240 - cy;
  putpixel((cx+x), (cy-y), color);
  putpixel((cx-x), (cy-y), color);
  putpixel((cx+x), (cy+y), color);
  putpixel((cx-x), (cy+y), color);
}

void bresHyperbola(int cx, int cy, int a,int b, int color)
{
  int d = (2*a*a) - (b*b) - (2*a*b*b);
  int x=a, y=0;
  while((a*a*y) <= (b*b*x))
  {
    if(d<= 0)
      d+= 2*a*a*(2*y+3);
    else {
      d+= 2*a*a*(2*y+3) - 4*b*b*(x+1);
      x++;
    }
    y++;
    plotHyperbola(cx, cy, x, y, color);
  }

 d = 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(x<220)
  {
    if(d<= 0) {
      d+= a*a*4*(y+1) - 2-a*a*b*b*(2*x+3)*(2*x+3);
      y++;
```

```
        }
        else
           d+= -2*b*b*a*a*(2*x+3);
        x++;
        plotHyperbola(cx, cy, x, y, color);
  }
}
```

Output:

```
int main(void){
   initwindow(640, 480, "Varun Sharma - 557/IC/13");
   drawAxes();
bresHyperbola(0, 0, 30, 50, WHITE);
}
```



Program 7

Objective:

To demonstrate a waving flag.

Code:

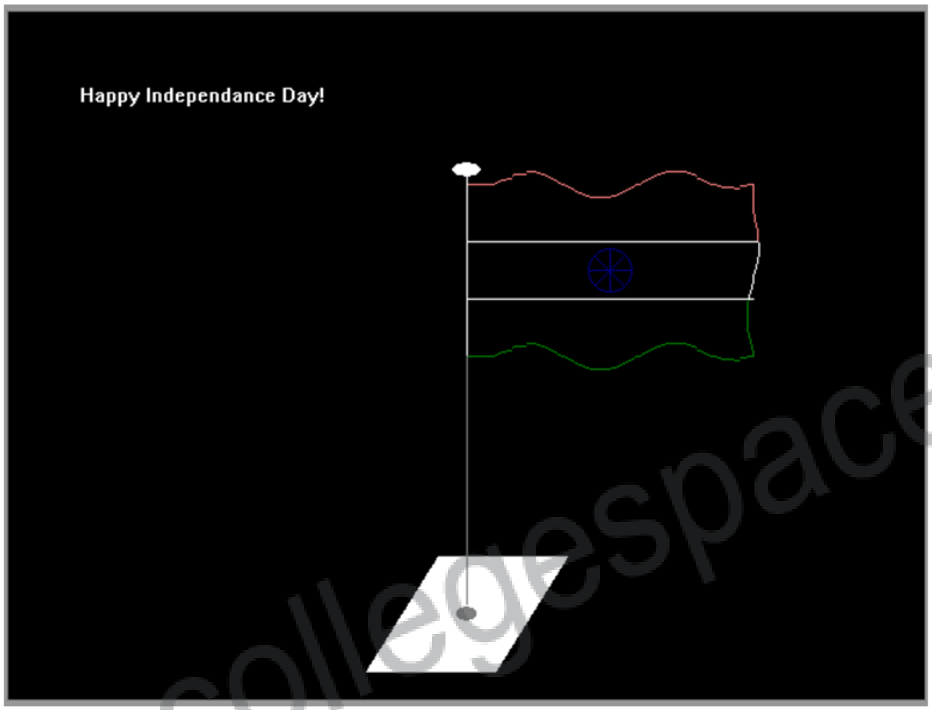
```
#include "midLine381.h"
#include "midCircle381.h"
#include "midEllipse381.h"
#include <graphics.h>
#include <math.h>

void flutter(void);
int upperPrev[406];
int upper[406];
int lowerPrev[406];
int lower[406];
int upperPoints;
int lowerPoints;

int main(void)
{
   initwindow(640, 480, "Varun Sharma - 557/IC/13");

   midLine(0, 0, 0, 130, WHITE);

   midLine(0, 40, 200, 40, WHITE);
   midLine(0, 80, 200, 80, WHITE);

   fillellipse(320, 240-130, 10, 5); //Upper Ellipse
   int r = 15;
   midCircle(100, 60, r, BLUE);
   midLine(100, 60-r, 100, 60+r, BLUE);
   midLine(100-r, 60, 100+r, 60, BLUE);
   midLine(100-r*sin(45)+3, 60-r*sin(45)+3, 100+r*sin(45)-3, 60+r*sin(45)-3, BLUE);
   midLine(100-r*sin(45)+3, 60+r*sin(45)-3, 100+r*sin(45)-3, 60-r*sin(45)+3, BLUE);

   midLine(-70, -220, -20, -140, WHITE);
   midLine(-20, -140, 70, -140, WHITE);
   midLine(20, -220, 70, -140, WHITE);
   midLine(-70, -220, 20, -220, WHITE);
   int podiumPoints[] = {-70+320, 240+220, 320-20, 240+140, 320+70, 240+140, 320+20, 240+220, 320-70, 240+220};
   fillpoly(5, podiumPoints);

   midLine(0, 0, 0, -180, DARKGRAY); //Pole
   setfillstyle(SOLID_FILL, DARKGRAY);
```

```
  fillellipse(320, 240+180, 8, 6); //Lower Ellipse

  outtextxy(50, 50, "Happy Independance Day!");
  flutter();
  while(!kbhit()) {}
  return 0;
}

void flutter(void)
{
  int prvLen1 = 0, prvLen2 = 0;
  int ypos = 0, xpos = 0;
  int winv = 30;  //inverse of frequency
  int amp = 0;
  int tprev = 0, winvprev = winv;
  float k = 0.06;
  int t = 0;int flagw = 1;int upperPointsPrev = 0;
  while (!kbhit())
  {
    //varying wind speed
    /*winvprev = winv;
    if (t%200)
    {
      if (t%40 == 0)
        winv += flagw;
    }
    else if (t%200 == 0)
    {
      flagw = -1*flagw;
    }*/
    /*winvprev = winv;
    if (t%50 == 0)
    {
      winv += -2;
    }
    if (t%250 == 0)
    {
      winv += 12;
    }*/

    t+=5;
    upperPoints = 0;lowerPoints = 0;
    for (int x=0; x<200; x++)
    {
      //To vary amplitude
      if (x <= 40)
      {
        if (x<=5)
          amp = 0;
        else if (x <= 10)
            amp = 2;
        else if (x <= 20)
            amp = 4;
        else if (x <= 30)
            amp = 6;
        else if (x <= 40)
            amp = 8;
    }
    else if (x >= 160)
    {
        if (x>=195)
            amp = 0;
        else if (x >= 190)
            amp = 2;
        else if (x >= 180)
            amp = 4;
        else if (x >= 170)
            amp = 6;
        else if (x >= 160)
            amp = 8;
    }
    else
        amp = 10;

    //Standing wave
    //ypos = 2*7*cos((tprev)/40)*sin(k*x);
    //xpos = 2*4*cos(tprev/40)*sin(k*x);

    //Travelling wave
    ypos = amp*sin(k*x - (float)tprev/winvprev);
    xpos = (amp/2)*sin(0.06*x - (float)tprev/winvprev);
    putpixel(320+x+1, 240-(120+ypos), BLACK);
    putpixel(320+x+1, 240-(0+ypos), BLACK);
    if (x < 121)
        putpixel(320+200+xpos, 120+x, BLACK);
}
for (int x=0; x<200; x++)
{
    //To vary amplitude
    if (x <= 40)
    {
        if (x<=5)
            amp = 0;
        else if (x <= 10)
            amp = 2;
        else if (x <= 20)
            amp = 4;
        else if (x <= 30)
```

```
            amp = 6;
        else if (x <= 40)
            amp = 8;
    }
    else if (x >= 160)
    {
        if (x>=195)
            amp = 0;
        else if (x >= 190)
            amp = 2;
        else if (x >= 180)
            amp = 4;
        else if (x >= 170)
            amp = 6;
        else if (x >= 160)
            amp = 8;
    }
    else
        amp = 10;

    //Standing wave
    //ypos = 2*7*cos(t/40)*sin(k*x);
    //xpos = 2*4*cos(t/40)*sin(k*x);

    //Travelling wave
    ypos = amp*sin(k*x - (float)t/winv);
    xpos = (amp/2)*sin(0.06*x - (float)t/winv);
    putpixel(320+x+1, 240-(120+ypos), LIGHTRED);
    putpixel(320+x+1, 240-(0+ypos), GREEN);
    if (x < 41)
        putpixel(320+200+xpos, 120+x, LIGHTRED);
    else if (x < 81)
        putpixel(320+200+xpos, 120+x, WHITE);
    else if (x < 121)
        putpixel(320+200+xpos, 120+x, GREEN);

    if (x == 40){
        if (prvLen1)
            midLine(0, 40, 200-prvLen1, 40, BLACK);
        midLine(0, 40, 200-xpos, 40, WHITE);
        prvLen1 = xpos;
    }
    else if (x == 80){
        if (prvLen2)
            midLine(0, 80, 200-prvLen2, 80, BLACK);
        midLine(0, 80, 200-xpos, 80, WHITE);
        prvLen2 = xpos;
    }
}
```

```
        tprev = t;
    }
}
```

Output:



Program 8

Objective:

To perform line clipping using different line clipping algorithms.

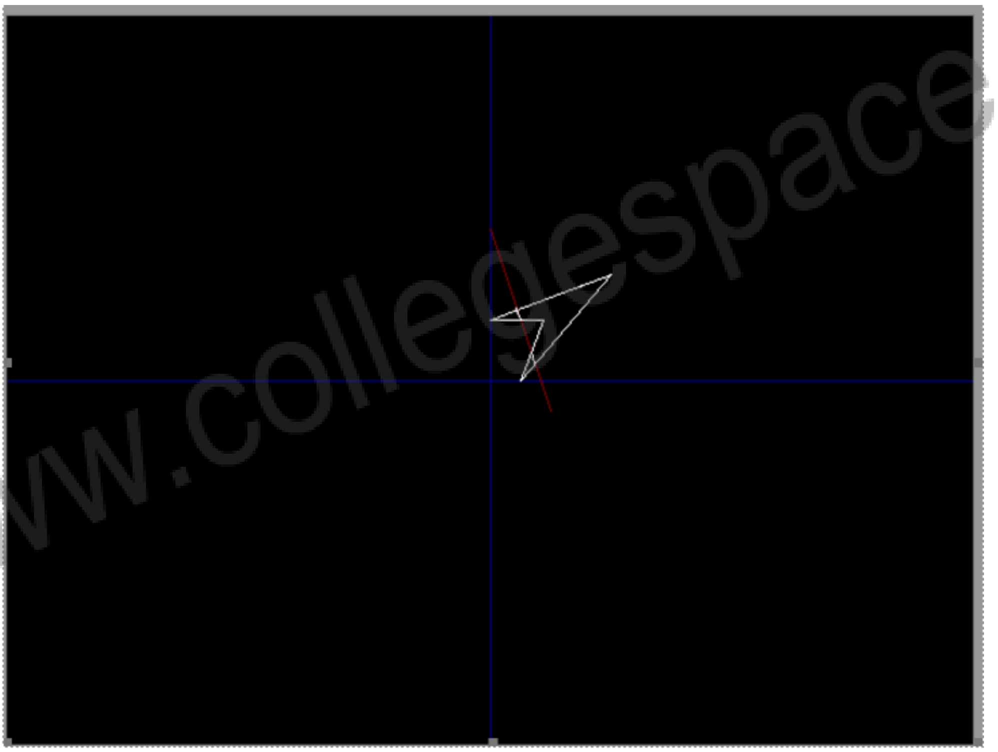
1. Using Cyrus-Beck line clipping algorithm -

Code:

```
#include <graphics.h>
#include "midLine381.h"

void drawAxes()
{
   midLine(-320, 0, 320, 0, BLUE);
   midLine(0, -240, 0, 240, BLUE);
}

int main()
{
   float t1, t2, t3, t4, x1, x2, x3, x4, y1, y2, y3, y4;
   float p1=0.0,p2=100.0,p3=40.0,p4=-20.0;
   initwindow(640, 480, "Varun Sharma - 557/IC/13");
   drawAxes();
   line(320+p1, 240-p2, 320+p3, 240-p4);
   //delay(2000);
   line(320+80,240-70,320+20,240-0);
   line(320+20,240-0,320+35,240-40);
   line(320+35,240-40,320+0,240-40);
   line(320+0,240-40,320+80,240-70);
   delay(50);
   t1=0.74;
   t2=0.68;
   t3=0.5;
   t4=0.44;
   x1=p1+p3*t1;
   y1=p2+(p4-p2)*t1;
   x2=p1+p3*t2;
   y2=p2+(p4-p2)*t2;
   x3=p1+p3*t3;
   y3=p2+(p4-p2)*t3;
   x4=p1+p3*t4;
```

```
    y4=p2+(p4-p2)*t4;
    setcolor(RED);
    line(320+p1, 240-p2, 320+p3, 240-p4);
    setcolor(WHITE);
    line(320+x1, 240-y1, 320+x2, 240-y2);
    line(320+x3, 240-y3, 320+x4, 240-y4);
    while (!kbhit()) {}
    closegraph();
    return 0;
}
```

Output:



2. UsingMid-PointSubdivision approach -

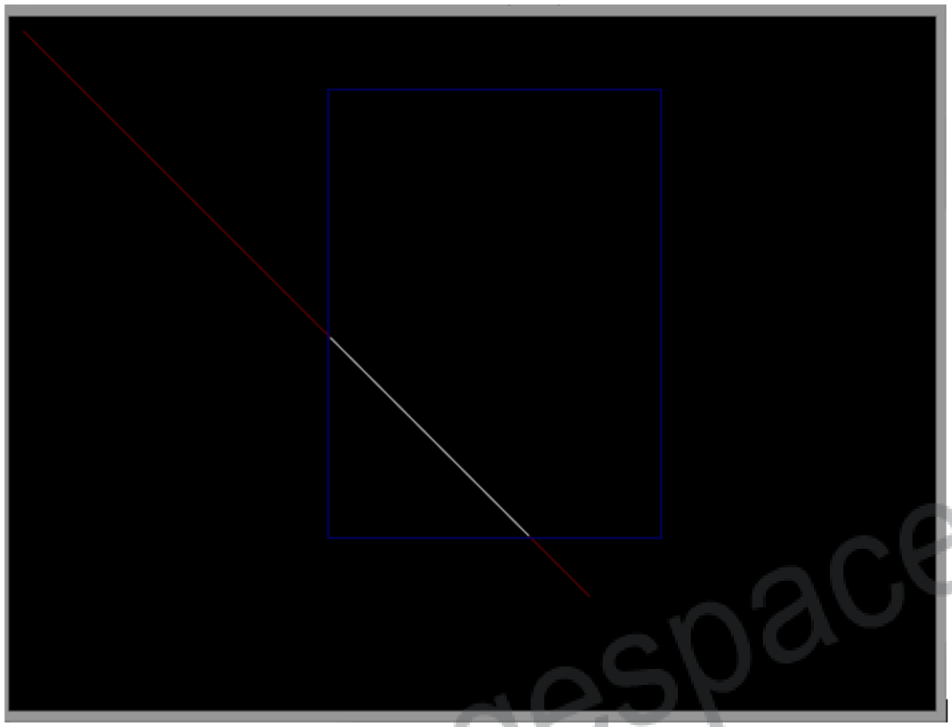
Code:

```
#include<cstdio>
#include<graphics.h>
#include<conio.h>
#include"lineutil.h"
#define ROUND(a) ((int)(a+0.5))
```

```
#define LEFT_EDGE 0x1
#define RIGHT_EDGE 0x2
#define BOTTOM_EDGE 0x4
#define TOP_EDGE 0x8
#define INSIDE(a) (!a)
#define REJECT(a,b) (a&b)
#define ACCEPT(a,b) (!(a|b))

struct window
{
  float x;
  float y;
};

struct point
{
  float x;
  float y;
};
unsigned char encode(point pt, window winMin,window winMax)
{
  unsigned char code=0x00;
  if(pt.x < winMin.x)
    code = code | LEFT_EDGE;
  if(pt.x > winMax.x)
    code = code | RIGHT_EDGE;
  if(pt.y < winMin.y)
    code = code | BOTTOM_EDGE;
  if(pt.y > winMax.y)
    code = code | TOP_EDGE;
  return(code);
}

void clipLineMid(window winMin, window winMax,point p1,point p2)
{
  struct point mid;
  int v;
  unsigned char code1,code2;
  code1=encode(p1,winMin,winMax);
  code2=encode(p2,winMin,winMax);
  if(ACCEPT(code1,code2))
    v=0;
  else if(REJECT(code1,code2))
```

```
        v=1;
    else
        v=2;

    switch(v)
    {
        case 0:
            p1.x=(int)p1.x;
            p1.y=(int)p1.y;
            p2.x=(int)p2.x;
            p2.y=(int)p2.y;
            /* Line conpletely visible */
            setcolor(WHITE);
            line(p1.x, p1.y, p2.x, p2.y);
        break;
        case 1:  /* Line completely invisible */
        break;
        case 2:  /* line partly visible */
            mid.x = p1.x + (p2.x-p1.x)/2;
            mid.y = p1.y + (p2.y-p1.y)/2;
            clipLineMid(winMin,winMax,p1,mid);
            mid.x = mid.x+1;
            mid.y = mid.y+1;
            clipLineMid(winMin,winMax,mid,p2);
        break;
    }
}
int main()
{
    initwindow(640, 480, "Varun Sharma - 557/IC/13");
    struct point p1, p2;struct window winMin, winMax;
    p1.x= 10.0;p1.y= 10.0;p2.x= 400.0;p2.y= 400.0;
    winMin.x=220.0;winMin.y=50.0;winMax.y=360.0;winMax.x=450.0;
    setcolor(RED);line(p1.x, p1.y, p2.x, p2.y);setcolor(BLUE);
    line(winMin.x, winMin.y, winMin.x, winMax.y);
    line(winMin.x, winMin.y, winMax.x, winMin.y);
    line(winMax.x, winMax.y, winMin.x, winMax.y);
    line(winMax.x, winMax.y, winMax.x, winMin.y);
    clipLineMid(winMin,winMax,p1,p2);
    while (!kbhit()) {}
    return 0;
}
```

Output:



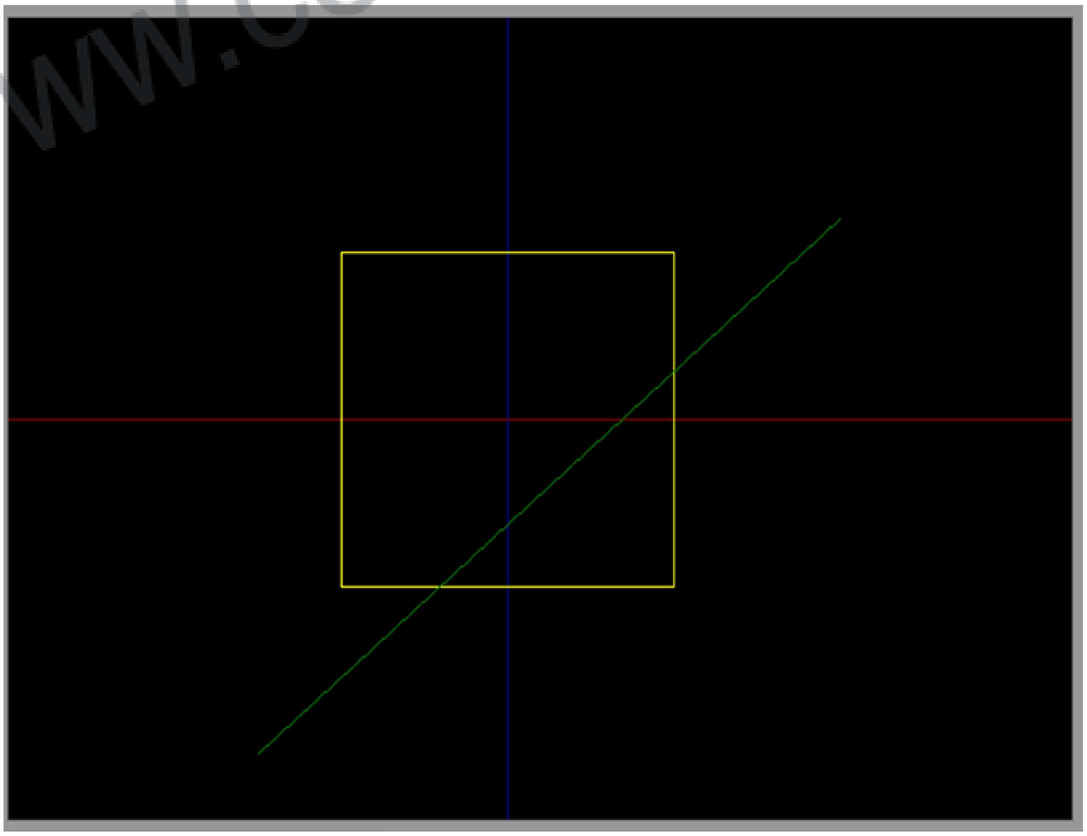
3. Using Cohen-Sutherland Approach -

Code:

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

using namespace std;

#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 clipLine (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;

   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);
```

```
}

int main()
{
   initwindow(640, 480, "Varun Sharma - 557/IC/13");
   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;

   clipLine(x1+300,240-y1,x2+300,240-y2,xwmin+300,240-ywmin,xwmax+300,240-ywmax);
   while(!kbhit()) {}
   closegraph();
}
```

Output:



4. Using Liang Barskey Approach -

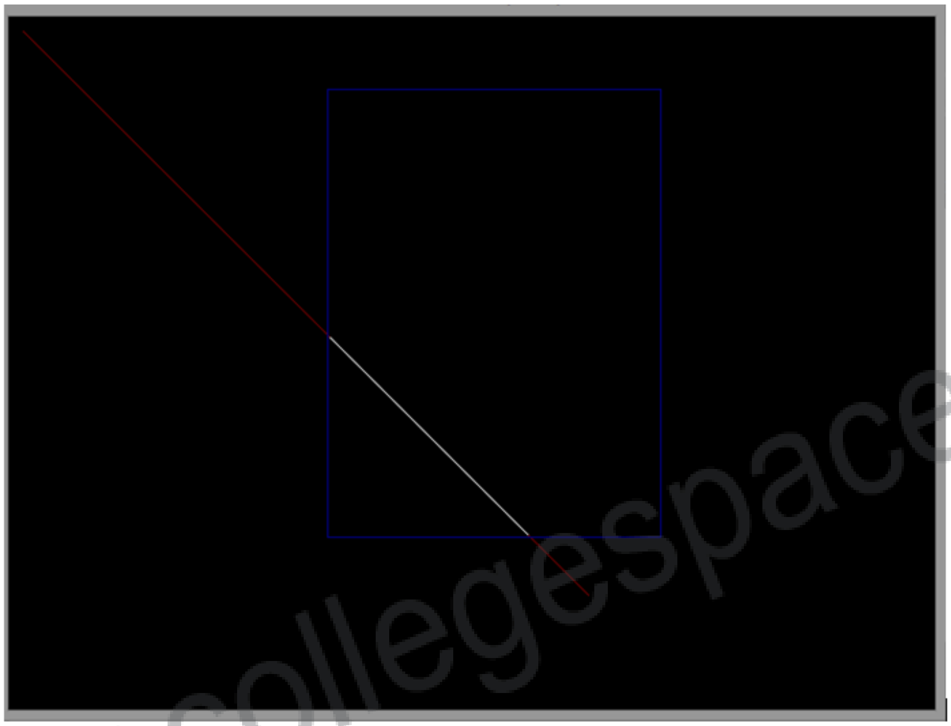
Code:

```
#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:



4. Using Nicholl-Lee-NichollApproach -

Code:

```
# 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)
    {
```

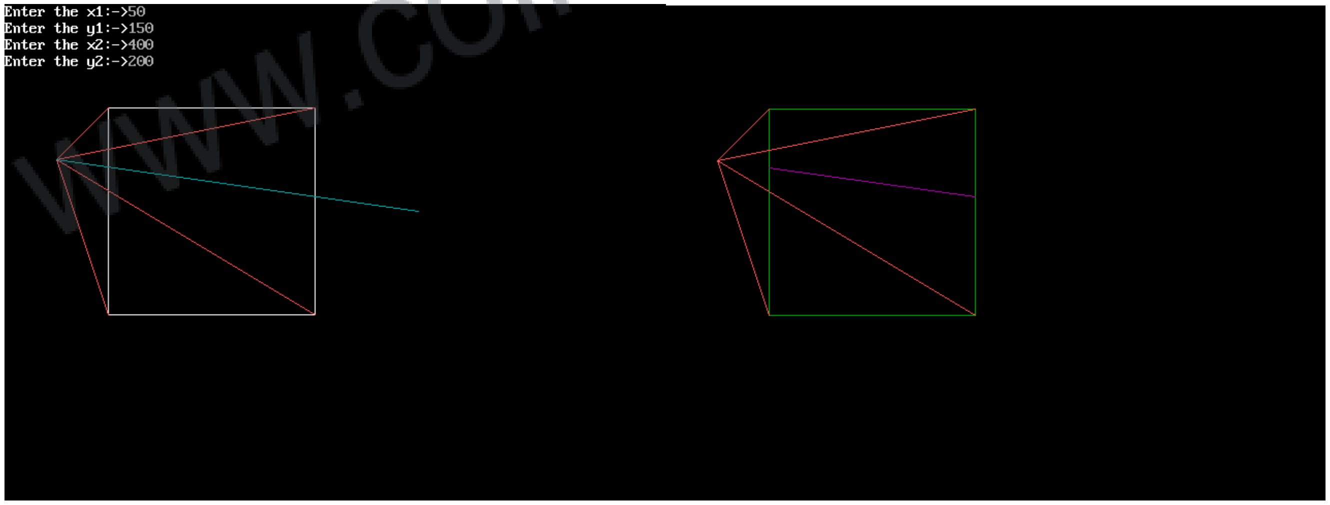
```
   nx2=x2;
   ny2=y2;
  }
  // point p2 is outside clip window
  else
  {
   ny2=ymin;
   nx2=x1+(ymin-y1)/m;
  }
 }
 // point p2 is on right side of clip window
 else if(m>m2 && m<m3 && x2>=x1)
 { // point p2 is inside clip window
  if(x2<xmax)
  {
   nx2=x2;
   ny2=y2;
  }
  // point p2 is outside 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 x1:->50
Enter the y1:->150
Enter the x2:->400
Enter the y2:->200
```

Program 9

Objective:

To perform polygon clipping using different techniques.

1. Using Sutherland-Hodgeman Approach -

Code:

```
#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
```

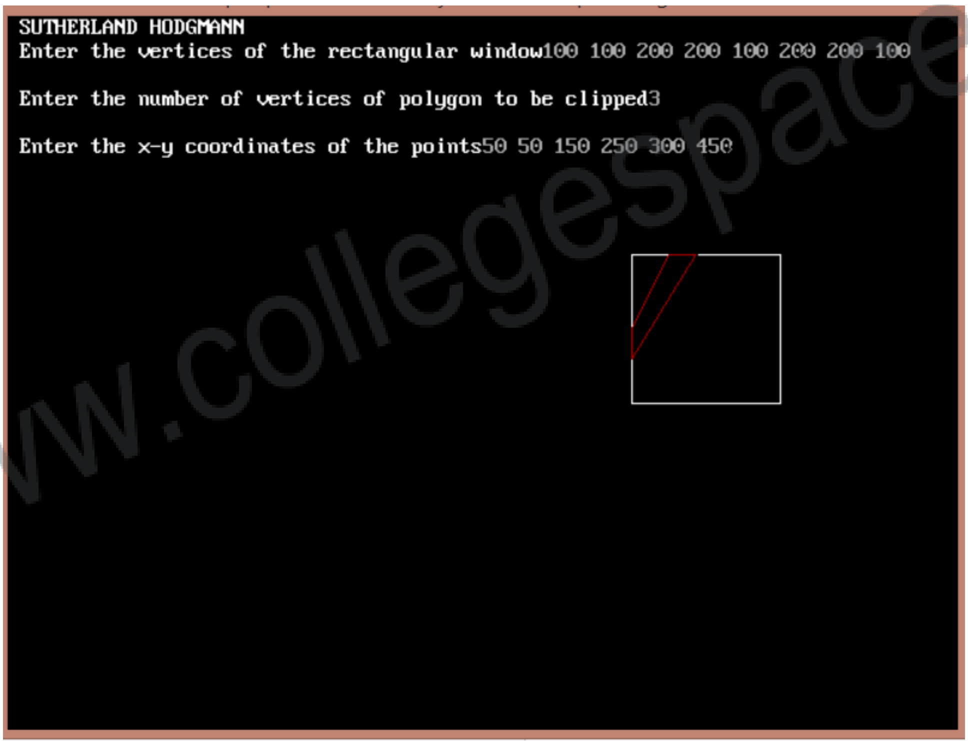
```
            int ay = int(((cl[0].x-s[j].x)*(s[j+1].y-s[j].y)/(s[j+1].x-s[j].x)*1.0)+s[j].y);
            t[k].x = cl[0].x;
            t[k].y = ay;
            k++;
            t[k] = s[j+1];
            k++;
      }
      //i -> o
      else if(s[j].x>cl[0].x && s[j+1].x<cl[0].x)
      {
            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:



```
SUTHERLAND HODGMANN
Enter the vertices of the rectangular window100 100 200 200 100 200 200 100

Enter the number of vertices of polygon to be clipped3

Enter the x-y coordinates of the points50 50 150 250 300 450
```

2. Using Weiler Atherton Approach -

Code:

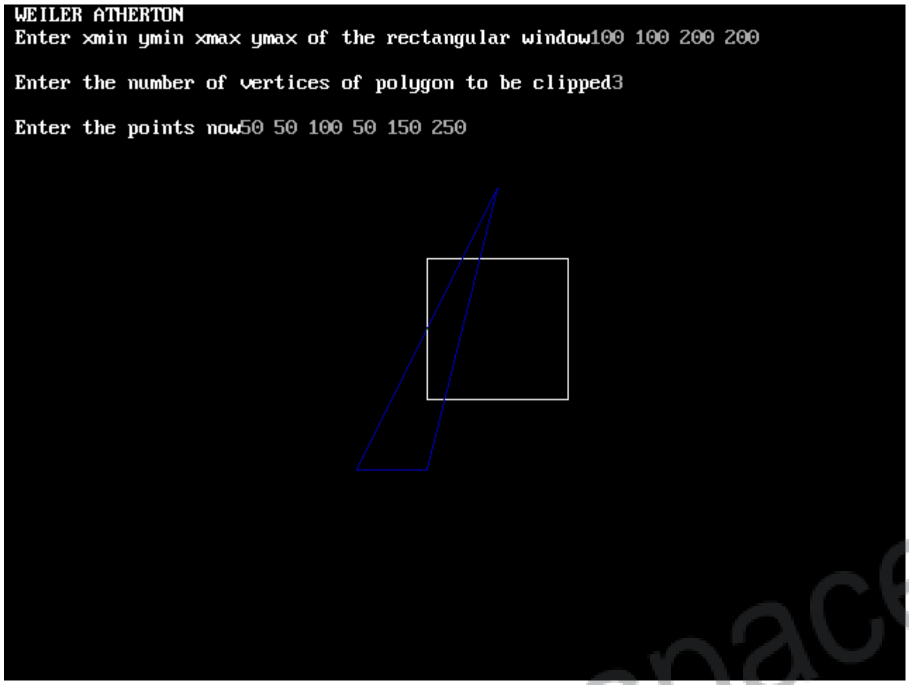
```
#include<graphics.h>
#include<iostream.h>
#include<conio.h>
#include<stdio.h>
#include<stdlib.h>
#include<dos.h>

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:



```
WEILER ATHERTON
Enter xmin ymin xmax ymax of the rectangular window100 100 200 200

Enter the number of vertices of polygon to be clipped3

Enter the points now50 50 100 50 150 250
```

Program 10

Objective:

To perform polygon filling using different techniques.

1. Using Scanline Approach –

Code:

```
#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;
        }
```

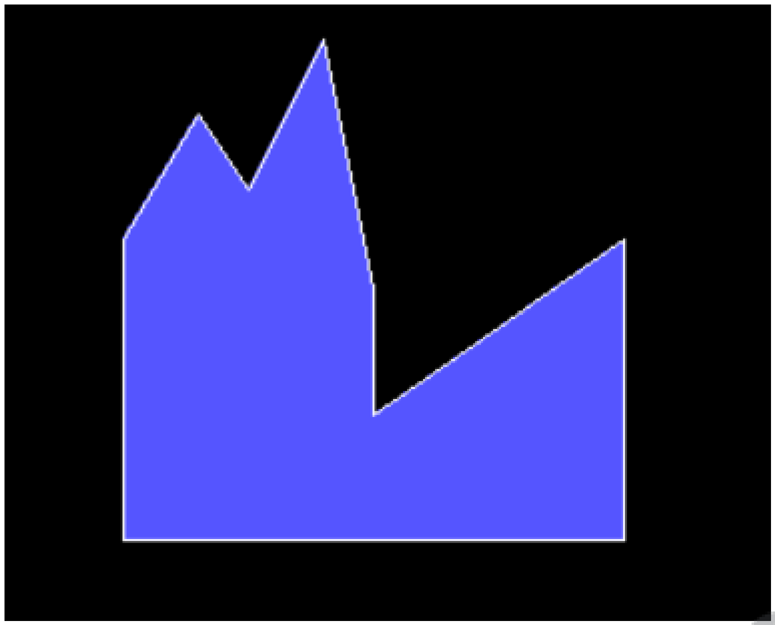
```
}
int linked_list_GET::search(node* item,node* &parent,node* &child)
{
        //parent follows the child
        child=head;
        while(child!=NULL)
        {
                if(child->ymax==item->ymax && child->x==item->x && child->delx==item->delx)
                        return 1;
                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:



2. Using Seed-Fill Approach –

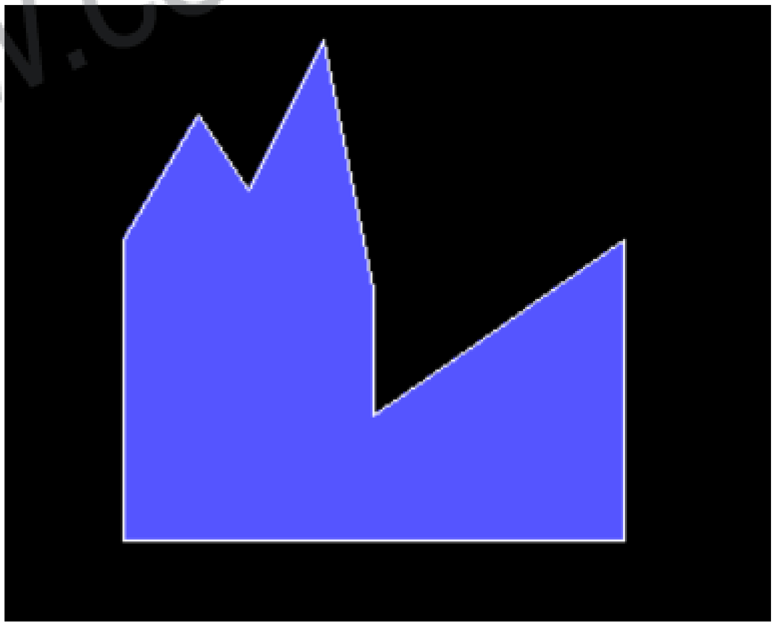
Code:

```
#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:



Program 11

Objective:
To perform 3D Transformations on a frustum and show different views.

Code:
```
#include <stdio.h>
#include <conio.h>
#include <math.h>
#include <graphics.h>
struct matrix
{
    int rows;
    int columns;
    double theMatrix[4][10];
};
struct vectorDimensions
{
    int x;
    int y;
    int z;
};
void plotFrustum(struct matrix,struct matrix,struct matrix);
void combineFrustums(struct matrix,struct matrix,struct matrix);
double radianize(int);
struct matrix Rotatex(struct matrix,struct matrix,int);
struct matrix Rotatey(struct matrix,struct matrix,int);
struct matrix Rotatez(struct matrix,struct matrix,int);
struct matrix compositeMatrixMultiply(struct matrix,struct matrix);
struct matrix initializeFrustum(struct matrix);
struct matrix perspectiveProjection(struct matrix,struct matrix,struct vectorDimensions,struct vectorDimensions,struct
vectorDimensions);
int main()
{
  initwindow(1280,720, "Varun Sharma - 557/IC/13");
  cleardevice();
  struct matrix frustum,translate,rotatex,rotatey,rotatez,perspective,frustumPerspective,frustumIsometric;
  struct vectorDimensions normal,pointOnPlane,centreOfProjection,lightSource;
  int i,j,dx,dy,dz,thetax,thetay,thetaz,theta;
  char dir1,dir2;
```

```
frustum = initializeFrustum(frustum);
frustumIsometric = initializeFrustum(frustumIsometric);
//Initialising the Rotate Matrices
rotatex.columns=4;
rotatex.rows=4;
rotatey.columns=4;
rotatey.rows=4;
rotatez.columns=4;
rotatez.rows=4;
for(i=0;i<4;i++)
{
 for(j=0;j<4;j++)
 {
  if(i==j)
  {
    rotatex.theMatrix[i][j]=1;
    rotatey.theMatrix[i][j]=1;
    rotatez.theMatrix[i][j]=1;
  }
  else
  {
    rotatex.theMatrix[i][j]=0;
    rotatey.theMatrix[i][j]=0;
    rotatez.theMatrix[i][j]=0;
  }
 }
}
//Plane for Perspective Projections
normal.x=1;
normal.y=1;
normal.z=0;
//Point on Plane
pointOnPlane.x=0;
pointOnPlane.y=0;
pointOnPlane.z=0;
//Centre of Projection
centreOfProjection.x=90;
centreOfProjection.y=30;
centreOfProjection.z=30;
//Storing the Perspective Projections
frustumPerspective = perspectiveProjection(perspective,frustum,normal,pointOnPlane,centreOfProjection);
//Isometric Projections
frustumIsometric = Rotatey(frustum,rotatey,45);
```

```
    frustumIsometric = Rotatex(frustumIsometric,rotatex,35);
    //Plotting the frustum's projections
    plotFrustum(frustum,frustumPerspective,frustumIsometric);
    combineFrustums(frustum,frustumPerspective,frustumIsometric);

    outtextxy(10, 10, "Front view"); outtextxy(500, 10, "Top view");
    outtextxy(10, 200, "Side view"); outtextxy(500, 200, "Isometric view");
    outtextxy(900, 100, "Perspective view");
    for(i=0;i<1000;i++)
    {
            //Rotation
            thetax=0; thetay=0; thetaz=0;
            dir1=getche();
            switch(dir1)
            {
                case 'y':
                  thetay += 5; break;
                case 'x':
                  thetax += 5; break;
                case 'z':
                  thetaz+=5; break;
            }
            if(thetax!=0)
               frustum = Rotatex(frustum,rotatex,thetax);
            if(thetay!=0)
               frustum = Rotatey(frustum,rotatey,thetay);
            if(thetaz!=0)
               frustum = Rotatez(frustum,rotatez,thetaz);
            frustumPerspective =
perspectiveProjection(perspective,frustum,normal,pointOnPlane,centreOfProjection);
            frustumIsometric = Rotatey(frustum,rotatey,45);
            frustumIsometric = Rotatex(frustumIsometric,rotatex,35);
            //Plotting After Rotation
            plotFrustum(frustum,frustumPerspective,frustumIsometric);
            combineFrustums(frustum,frustumPerspective,frustumIsometric);
            outtextxy(10, 10, "Front view");
            outtextxy(500, 10, "Top view");
            outtextxy(10, 200, "Side view");
            outtextxy(500, 200, "Isometric view");
            outtextxy(900, 100, "Perspective view");
    }
    getch();
    return 0;
```

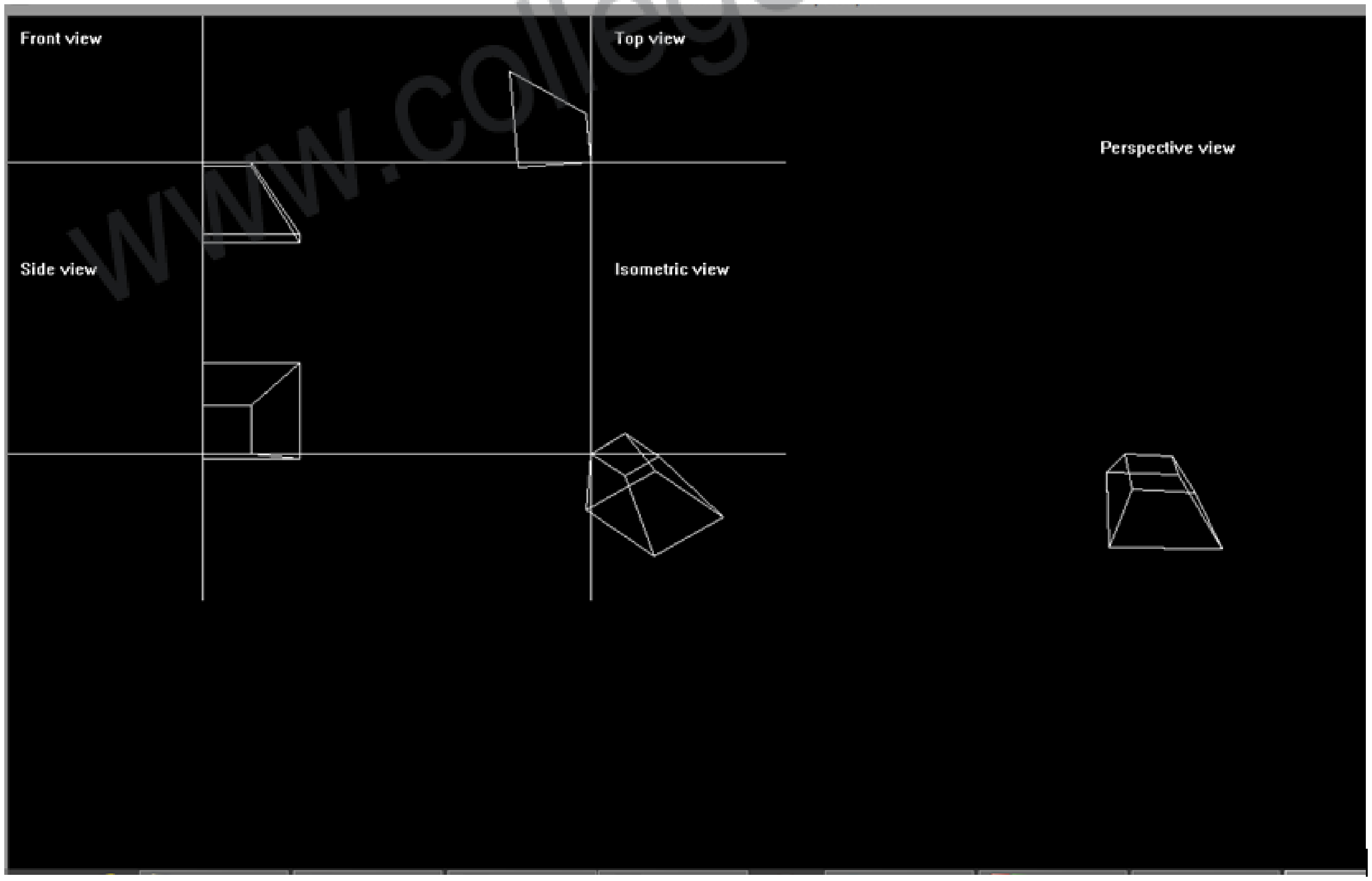
```
}
void plotFrustum(struct matrix frustum,struct matrix frustumPerspective, struct matrix frustumIsometric)
{
   cleardevice();
   int i;
   //Plot on X-Y
   line(0,120,640,120); line(160,0,160,480); line(0,360,640,360); line(480,0,480,480);
   for(i=0;i<frustum.columns;i++)
   {
    //X-Y
    putpixel((int)(160+frustum.theMatrix[0][i]),(int)(120-frustum.theMatrix[1][i]),3);
    //Y-Z
    putpixel((int)(480+frustum.theMatrix[1][i]),(int)(120-frustum.theMatrix[2][i]),3);
    //X-Z
    putpixel((int)(160+frustum.theMatrix[0][i]),(int)(360-frustum.theMatrix[2][i]),3);
    //Perspective
    putpixel((int)(920+frustumPerspective.theMatrix[2][i]),(int)(360-frustumPerspective.theMatrix[1][i]),3);
    //Isometric
    putpixel((int)(480+frustumIsometric.theMatrix[0][i]),(int)(360-frustumIsometric.theMatrix[1][i]),3);
   }
}
void combineFrustums(struct matrix frustum,struct matrix frustumPerspective,struct matrix frustumIsometric)
{
    int i;
    for(i=0;i<7;i++)
    {
     if(i!=3)
     {
       //X-Y
       line((int)(160+frustum.theMatrix[0][i]),(int)(120-
frustum.theMatrix[1][i]),(int)(160+frustum.theMatrix[0][i+1]),(int)(120-frustum.theMatrix[1][i+1]));
       //Y-Z
       line((int)(480+frustum.theMatrix[1][i]),(int)(120-
frustum.theMatrix[2][i]),(int)(480+frustum.theMatrix[1][i+1]),(int)(120-frustum.theMatrix[2][i+1]));
       //X-Z
       line((int)(160+frustum.theMatrix[0][i]),(int)(360-
frustum.theMatrix[2][i]),(int)(160+frustum.theMatrix[0][i+1]),(int)(360-frustum.theMatrix[2][i+1]));
       //Perspective
       line((int)(920+frustumPerspective.theMatrix[2][i]),(int)(360-
frustumPerspective.theMatrix[1][i]),(int)(920+frustumPerspective.theMatrix[2][i+1]),(int)(360-
frustumPerspective.theMatrix[1][i+1]));
       //Isometric
```

```
        line((int)(480+frustumIsometric.theMatrix[0][i]),(int)(360-
frustumIsometric.theMatrix[1][i]),(int)(480+frustumIsometric.theMatrix[0][i+1]),(int)(360-
frustumIsometric.theMatrix[1][i+1]));
      }
    }
    //X-Y
    line((int)(160+frustum.theMatrix[0][0]),(int)(120-
frustum.theMatrix[1][0]),(int)(160+frustum.theMatrix[0][3]),(int)(120-frustum.theMatrix[1][3]));
    line((int)(160+frustum.theMatrix[0][4]),(int)(120-
frustum.theMatrix[1][4]),(int)(160+frustum.theMatrix[0][7]),(int)(120-frustum.theMatrix[1][7]));
    //Y-Z
    line((int)(480+frustum.theMatrix[1][0]),(int)(120-
frustum.theMatrix[2][0]),(int)(480+frustum.theMatrix[1][3]),(int)(120-frustum.theMatrix[2][3]));
    line((int)(480+frustum.theMatrix[1][4]),(int)(120-
frustum.theMatrix[2][4]),(int)(480+frustum.theMatrix[1][7]),(int)(120-frustum.theMatrix[2][7]));
    //X-Z
    line((int)(160+frustum.theMatrix[0][0]),(int)(360-
frustum.theMatrix[2][0]),(int)(160+frustum.theMatrix[0][3]),(int)(360-frustum.theMatrix[2][3]));
    line((int)(160+frustum.theMatrix[0][4]),(int)(360-
frustum.theMatrix[2][4]),(int)(160+frustum.theMatrix[0][7]),(int)(360-frustum.theMatrix[2][7]));
    //Perspective
    line((int)(920+frustumPerspective.theMatrix[2][0]),(int)(360-
frustumPerspective.theMatrix[1][0]),(int)(920+frustumPerspective.theMatrix[2][3]),(int)(360-
frustumPerspective.theMatrix[1][3]));
    line((int)(920+frustumPerspective.theMatrix[2][4]),(int)(360-
frustumPerspective.theMatrix[1][4]),(int)(920+frustumPerspective.theMatrix[2][7]),(int)(360-
frustumPerspective.theMatrix[1][7]));
    //Isometric
    line((int)(480+frustumIsometric.theMatrix[0][0]),(int)(360-
frustumIsometric.theMatrix[1][0]),(int)(480+frustumIsometric.theMatrix[0][3]),(int)(360-
frustumIsometric.theMatrix[1][3]));
    line((int)(480+frustumIsometric.theMatrix[0][4]),(int)(360-
frustumIsometric.theMatrix[1][4]),(int)(480+frustumIsometric.theMatrix[0][7]),(int)(360-
frustumIsometric.theMatrix[1][7]));
    for(i=0;i<4;i++)
    {
      //X-Y
      line((int)(160+frustum.theMatrix[0][i]),(int)(120-
frustum.theMatrix[1][i]),(int)(160+frustum.theMatrix[0][i+4]),(int)(120-frustum.theMatrix[1][i+4]));
      //Y-Z
      line((int)(480+frustum.theMatrix[1][i]),(int)(120-
frustum.theMatrix[2][i]),(int)(480+frustum.theMatrix[1][i+4]),(int)(120-frustum.theMatrix[2][i+4]));
      //X-Z
    line((int)(160+frustum.theMatrix[0][i]),(int)(360-
frustum.theMatrix[2][i]),(int)(160+frustum.theMatrix[0][i+4]),(int)(360-frustum.theMatrix[2][i+4]));
    //Perspective
    line((int)(920+frustumPerspective.theMatrix[2][i]),(int)(360-
frustumPerspective.theMatrix[1][i]),(int)(920+frustumPerspective.theMatrix[2][i+4]),(int)(360-
frustumPerspective.theMatrix[1][i+4]));
    //Isometric
    line((int)(480+frustumIsometric.theMatrix[0][i]),(int)(360-
frustumIsometric.theMatrix[1][i]),(int)(480+frustumIsometric.theMatrix[0][i+4]),(int)(360-
frustumIsometric.theMatrix[1][i+4]));
  }
}
struct matrix Rotatex(struct matrix frustum,struct matrix rotatex,int theta)
{
    struct matrix temp;
    rotatex.theMatrix[1][1] = cos(radianize(theta));
    rotatex.theMatrix[1][2] = -sin(radianize(theta));
    rotatex.theMatrix[2][1] = sin(radianize(theta));
    rotatex.theMatrix[2][2] = cos(radianize(theta));
    temp = compositeMatrixMultiply(rotatex,frustum);
    return temp;
}
struct matrix Rotatey(struct matrix frustum,struct matrix rotatey,int theta)
{
    struct matrix temp;
    rotatey.theMatrix[0][0] = cos(radianize(theta));
    rotatey.theMatrix[0][2] = sin(radianize(theta));
    rotatey.theMatrix[2][0] = -sin(radianize(theta));
    rotatey.theMatrix[2][2] = cos(radianize(theta));
    temp = compositeMatrixMultiply(rotatey,frustum);
    return temp;
}
struct matrix Rotatez(struct matrix frustum,struct matrix rotatez,int theta)
{
    struct matrix temp;
    rotatez.theMatrix[0][0] = cos(radianize(theta));
    rotatez.theMatrix[0][1] = -sin(radianize(theta));
    rotatez.theMatrix[1][0] = sin(radianize(theta));
    rotatez.theMatrix[1][1] = cos(radianize(theta));
    temp = compositeMatrixMultiply(rotatez,frustum);
    return temp;
}
double radianize(int theta) {
      return ((theta*3.14)/180);
}
struct matrix compositeMatrixMultiply(struct matrix composite,struct matrix frustum)
{
     int i,j,k;
     struct matrix temp;
     temp.rows = composite.rows; temp.columns = frustum.columns;
     for(i=0;i<composite.rows;i++)
     {
       for(j=0;j<frustum.columns;j++)
       {
         temp.theMatrix[i][j]=0;
         for(k=0;k<frustum.rows;k++)
         {
           temp.theMatrix[i][j]+=composite.theMatrix[i][k]*frustum.theMatrix[k][j];
         }
       }
     }
     return temp;
}

struct matrix initializeFrustum(struct matrix frustum)
{
    for(int i=0;i<8;i++)
          frustum.theMatrix[3][i]=1;
    //Point 1
    frustum.theMatrix[0][0]=0;
    frustum.theMatrix[1][0]=0;
    frustum.theMatrix[2][0]=0;
    //Point 2
    frustum.theMatrix[0][1]=0;
    frustum.theMatrix[1][1]=40;
    frustum.theMatrix[2][1]=0;
    //Point 3
    frustum.theMatrix[0][2]=40;
    frustum.theMatrix[1][2]=40;
    frustum.theMatrix[2][2]=0;
    //Point 4
    frustum.theMatrix[0][3]=40;
    frustum.theMatrix[1][3]=0;
    frustum.theMatrix[2][3]=0;
    //Point 5
    frustum.theMatrix[0][4]=0;
```

```
   frustum.theMatrix[1][4]=0;
   frustum.theMatrix[2][4]=60;
   //Point 6
   frustum.theMatrix[0][5]=0;
   frustum.theMatrix[1][5]=80;
   frustum.theMatrix[2][5]=60;
   //Point 7
   frustum.theMatrix[0][6]=80;
   frustum.theMatrix[1][6]=80;
   frustum.theMatrix[2][6]=60;
   //Point 8
   frustum.theMatrix[0][7]=80;
   frustum.theMatrix[1][7]=0;
   frustum.theMatrix[2][7]=60;
   frustum.rows=4;
   frustum.columns=8;
   return frustum;
}
struct matrix perspectiveProjection(struct matrix perspective,struct matrix frustum,struct vectorDimensions
normal,struct vectorDimensions pointOnPlane,struct vectorDimensions centreOfProjection)
{
     struct matrix temp;
     float D,Do,D1; int i,j;
     perspective.rows = 4; perspective.columns = 4;
     //Do = Xon1 + Yon2 + Zon3;
     Do = (float)((pointOnPlane.x * normal.x) + (pointOnPlane.y * normal.y) + (pointOnPlane.z * normal.z));
     //D1 = an1 + bn2 + cn3;
     D1 = (float)((centreOfProjection.x * normal.x) + (centreOfProjection.y * normal.y) + (centreOfProjection.z *
normal.z));
     D = Do-D1;
     //Initializing the Perspective Matrix
     perspective.theMatrix[0][0]=(float)((centreOfProjection.x * normal.x) + D);
     perspective.theMatrix[0][1]=(float)((centreOfProjection.x * normal.y));
     perspective.theMatrix[0][2]=(float)((centreOfProjection.x * normal.z));
     perspective.theMatrix[0][3]=(float)(-(centreOfProjection.x * Do));
     perspective.theMatrix[1][0]=(float)((centreOfProjection.y * normal.x));
     perspective.theMatrix[1][1]=(float)((centreOfProjection.y * normal.y) + D);
     perspective.theMatrix[1][2]=(float)((centreOfProjection.y * normal.z));
     perspective.theMatrix[1][3]=(float)(-(centreOfProjection.y * Do));
     perspective.theMatrix[2][0]=(float)((centreOfProjection.z * normal.x));
     perspective.theMatrix[2][1]=(float)((centreOfProjection.z * normal.y));
     perspective.theMatrix[2][2]=(float)((centreOfProjection.z * normal.z) + D);
     perspective.theMatrix[2][3]=(float)(-(centreOfProjection.z * Do));
```

```
      perspective.theMatrix[3][0]=(float)((normal.x));
      perspective.theMatrix[3][1]=(float)((normal.y));
      perspective.theMatrix[3][2]=(float)((normal.z));
      perspective.theMatrix[3][3]=(float)(-D1);
      //Sending for Matrix Multiplication
      temp = compositeMatrixMultiply(perspective,frustum);
      //Dividing by H
      for(i=0;i<3;i++)
      {
        for(j=0;j<temp.columns;j++)
        {
          temp.theMatrix[i][j] = ((temp.theMatrix[i][j])/(temp.theMatrix[3][j]));
        }
      }
      return temp;
}
```

Output:



Program 12

Objective:

To perform Hidden Surface elimination using Back Face Detection.

Code:

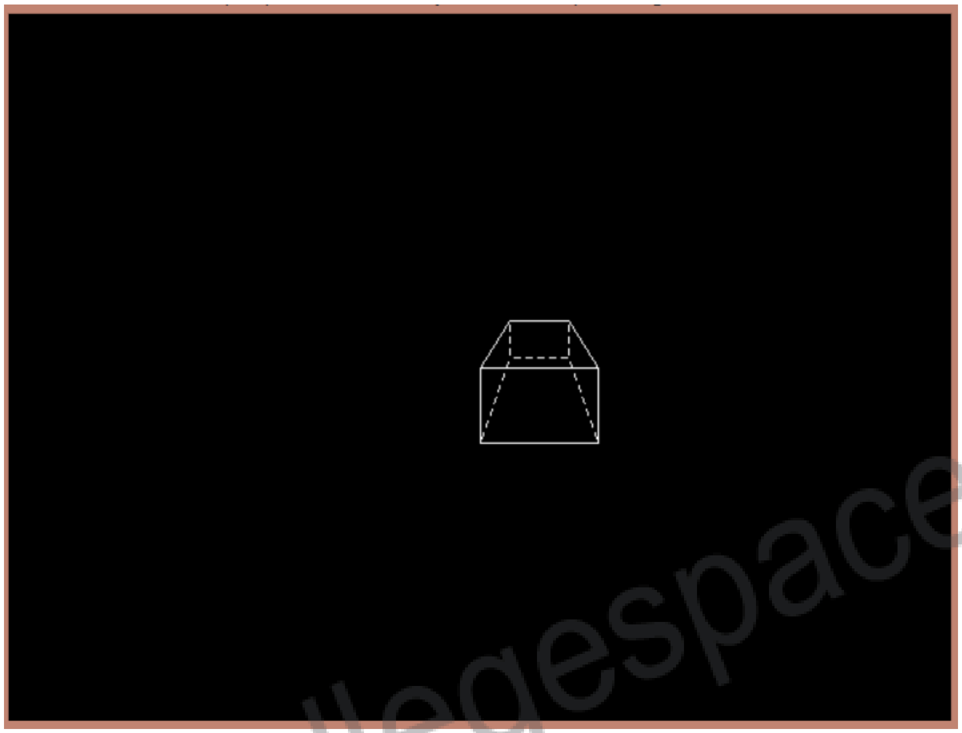
```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<graphics.h>
#include<stdlib.h>
#include<dos.h>
#define pi 3.14

int xf=520,xmax=640,ymax=480,view=1;
float xc=(xf-20)/2,yc=(ymax-40)/2;
char mindex='0';

float prism[4][8]={0 ,80,80,0 ,20,60,60,20,
                   80,80,0 ,0 ,60,60,20,20,
                   0 ,0 ,0 ,0 ,60,60,60,60,
                   1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 };


float dot(float n1[],float n2[])
{
        int i;
        float ans=0;
        for(i=0;i<3;i++)
        {
                ans+=(n1[i]*n2[i]);
        }
        return ans;
}
float mag(float n[])
{
        return(sqrt(n[0]*n[0]+n[1]*n[1]+n[2]*n[2]));
}

void normal(float n[],float a,float b,float c,float s1,float s2)
{
        int i;
        float d,n1[3],n2[3],n3[3],v[3],cos;
        for(i=0;i<3;i++)
        {
                n1[i]=prism[i][a];
                n2[i]=prism[i][b];
                n3[i]=prism[i][c];
                v[i]=prism[i][s1]-prism[i][s2];
        }
        n[0]=((n1[1]-n2[1])*(n2[2]-n3[2]))-((n1[2]-n2[2])*(n2[1]-n3[1]));
        n[1]=((n1[2]-n2[2])*(n2[0]-n3[0]))-((n1[0]-n2[0])*(n2[2]-n3[2]));
        n[2]=((n1[0]-n2[0])*(n2[1]-n3[1]))-((n1[1]-n2[1])*(n2[0]-n3[0]));
        d=n1[0]*(n2[1]*n3[2]-n3[1]*n2[2])-n1[1]*(n2[0]*n3[2]-n3[0]*n2[2])+n1[2]*(n2[0]*n3[1]-n3[0]*n2[1]);
        cos=dot(n,v)/(mag(n)*mag(v));
        if(cos>0)
        for(i=0;i<3;i++)
                n[i]=n[i]*(-1);
}

void mp(float n[],float a,float b)
{
        int i;
        for(i=0;i<3;i++)
                n[i]=(prism[i][a]+prism[i][b])/2;
        n[2]-=32768;
}

void fline(int x0,int y0,int x1,int y1,int x2,int y2,int c1=15,int style=0)
{
        setcolor(c1);
        setlinestyle(style,1,1);
        line(x0+x1,y0-y1,x0+x2,y0-y2);
        setcolor(15);
        setlinestyle(0,1,1);
}

void surface(int x0,int y0,int t,float prism[][8],int color[])
{
        int i,j,k,style=0;
        float n[3],n2[3];
        if(t==0)  //ABCD
        {
                normal(n,0,1,2,4,0);
                mp(n2,0,2);
                if(dot(n,n2)>0)
                        style=3;
                for(i=0;i<4;i++)
                {
                        j=i+1;
                        if(j==4)
                                j=0;
                        fline(x0,y0,prism[0][i],prism[1][i],prism[0][j],prism[1][j],color[t],style);

                }
                setfillstyle(SOLID_FILL,color[t]);


        }
        else if(t==1)       //EFGH
        {
                normal(n,4,5,6,0,4);
                mp(n2,4,6);
                if(dot(n,n2)>0)
                        style=3;
                for(i=4;i<8;i++)
                {
                        j=i+1;
                        if(j==8)
                                j=4;
                        fline(x0,y0,prism[0][i],prism[1][i],prism[0][j],prism[1][j],color[t],style);
                }
                setfillstyle(SOLID_FILL,color[t]);



        }
        else if(t==2)  //GFBC
        {
                normal(n,6,5,1,4,5);
                mp(n2,6,1);
                if(dot(n,n2)>0)
                        style=3;
                fline(x0,y0,prism[0][6],prism[1][6],prism[0][5],prism[1][5],color[t],style);//GF
                fline(x0,y0,prism[0][5],prism[1][5],prism[0][1],prism[1][1],color[t],style);  //FB
                fline(x0,y0,prism[0][1],prism[1][1],prism[0][2],prism[1][2],color[t],style);  //BC
                fline(x0,y0,prism[0][2],prism[1][2],prism[0][6],prism[1][6],color[t],style);  //CG
                setfillstyle(SOLID_FILL,color[t]);
```

```
}
else if(t==3)       //GCDH
{
            normal(n,6,2,3,5,6);
            mp(n2,6,3);
            if(dot(n,n2)>0)
                        style=3;
            fline(x0,y0,prism[0][6],prism[1][6],prism[0][2],prism[1][2],color[t],style);//GC
            fline(x0,y0,prism[0][2],prism[1][2],prism[0][3],prism[1][3],color[t],style);//CD
            fline(x0,y0,prism[0][3],prism[1][3],prism[0][7],prism[1][7],color[t],style);//DH
            fline(x0,y0,prism[0][7],prism[1][7],prism[0][6],prism[1][6],color[t],style);//HG
            setfillstyle(SOLID_FILL,color[t]);


}
else if(t==4)       //AEFB
{
            normal(n,4,5,1,6,5);
            mp(n2,1,5);
            if(dot(n,n2)>0)
                        style=3;
            fline(x0,y0,prism[0][0],prism[1][0],prism[0][4],prism[1][4],color[t],style);//AE
            fline(x0,y0,prism[0][4],prism[1][4],prism[0][5],prism[1][5],color[t],style);//EF
            fline(x0,y0,prism[0][5],prism[1][5],prism[0][1],prism[1][1],color[t],style);//FB
            fline(x0,y0,prism[0][1],prism[1][1],prism[0][0],prism[1][0],color[t],style);//BA
            setfillstyle(SOLID_FILL,color[t]);



}
else if(t==5)       //EADH
{
            normal(n,4,0,3,5,4);
            mp(n2,4,3);
            if(dot(n,n2)>0)
                        style=3;
            fline(x0,y0,prism[0][4],prism[1][4],prism[0][0],prism[1][0],color[t],style);//EA
            fline(x0,y0,prism[0][0],prism[1][0],prism[0][3],prism[1][3],color[t],style);//AD
            fline(x0,y0,prism[0][3],prism[1][3],prism[0][7],prism[1][7],color[t],style);//DH
            fline(x0,y0,prism[0][7],prism[1][7],prism[0][4],prism[1][4],color[t],style);//HE
            setfillstyle(SOLID_FILL,color[t]);
```

```
        }
}
void front(float prism[][8],int x0=320,int y0=240)
{
        int i,j;
        float n[6][3],M[6][3];
        int color[6]={15,15,15,15,15,15};
        for(i=0;i<6;i++)
                surface(x0,y0,i,prism,color);

}

void multiplymatrix(float a[][4],float b[][8],float c[][8])
{
        int i,j,k;
        for(i=0;i<4;i++)
        {
                for(j=0;j<8;j++)
                {
                        c[i][j]=0;
                        for(k=0;k<4;k++)
                                c[i][j]+=a[i][k]*b[k][j];
                }
        }
}

void rotation(float prism[][8],char ch,float A)
{
        float abcd[4][4],efgh[4][4],ABCD[4][4],EFGH[4][4],prism2[4][8];
        int i,j,k;
        float c[4][4];
        float a=A*pi/180;
        if(ch=='x')
        {
                for(i=0;i<4;i++)
                {
                        for(j=0;j<4;j++)
                        {
                                if(i==j)
                                        c[i][j]=1;
                                else
                                        c[i][j]=0;
                }
                for(j=0;j<4;j++)
                {
                        if(j==1)
                        {
                                if(i==1)
                                        c[i][j]=cos(a);
                                else if(i==2)
                                        c[i][j]=sin(a);
                        }
                        if(j==2)
                        {
                                if(i==1)
                                        c[i][j]=-1*sin(a);
                                if(i==2)
                                        c[i][j]=cos(a);
                        }
                }
        }
}
if(ch=='y')
{
        for(i=0;i<4;i++)
        {
                for(j=0;j<4;j++)
                {
                        if(i==j)
                                c[i][j]=1;
                        else
                                c[i][j]=0;
                }
                for(j=0;j<4;j++)
                {
                        if(j==0)
                        {
                                if(i==0)
                                        c[i][j]=cos(a);
                                else if(i==2)
                                        c[i][j]=-1*sin(a);
                        }
                        if(j==2)
                        {
                                if(i==0)
                                c[i][j]=sin(a);
                        if(i==2)
                                c[i][j]=cos(a);
                    }
                }
            }
        }
        if(ch=='z')
        {
            for(i=0;i<4;i++)
            {
                for(j=0;j<4;j++)
                {
                    if(i==j)
                        c[i][j]=1;
                    else
                        c[i][j]=0;
                }
                for(j=0;j<4;j++)
                {
                    if(j==0)
                    {
                        if(i==0)
                            c[i][j]=cos(a);
                        else if(i==1)
                            c[i][j]=sin(a);
                    }
                    if(j==1)
                    {
                        if(i==0)
                            c[i][j]=-1*sin(a);
                        if(i==1)
                            c[i][j]=cos(a);
                    }
                }
            }
        }
        multiplymatrix(c,prism,prism2);
        for(i=0;i<4;i++)
            for(j=0;j<8;j++)
                prism[i][j]=prism2[i][j];
}
```

```
void main()
{

	/* request auto detection */
   int gdriver = DETECT, gmode, errorcode;
   int xmax, ymax;
   char ch='x';
   float abcd[4][4],efgh[4][4],ABCD[4][4],EFGH[4][4],c[4][4];
	int i,j,k,m;
   /* initialize graphics and local variables */
   initgraph(&gdriver, &gmode, "c:\\TC\\BGI");

   /* read result of initialization */
   errorcode = graphresult();
   /* an error occurred */
   if (errorcode != grOk)
   {
      printf("Graphics error: %s\n", grapherrormsg(errorcode));
      printf("Press any key to halt:");
      getch();
      exit(1);
   }
	ch='x';
   while(1)
   {
	cleardevice();
	if(kbhit()!=0)
		ch=getch();
		if(ch=='e')
			exit(0);
		else if(ch=='x')
				rotation(prism,'x',2);
		else if(ch=='y')
				rotation(prism,'y',2);
		else if(ch=='z')
				rotation(prism,'z',2);

	front(prism);
	delay(50);
   }
	/* clean up */
   getch();
   closegraph();
  // return 0;
}
```

Output:



Program 13

Objective:

To demonstrate2D Transformations.

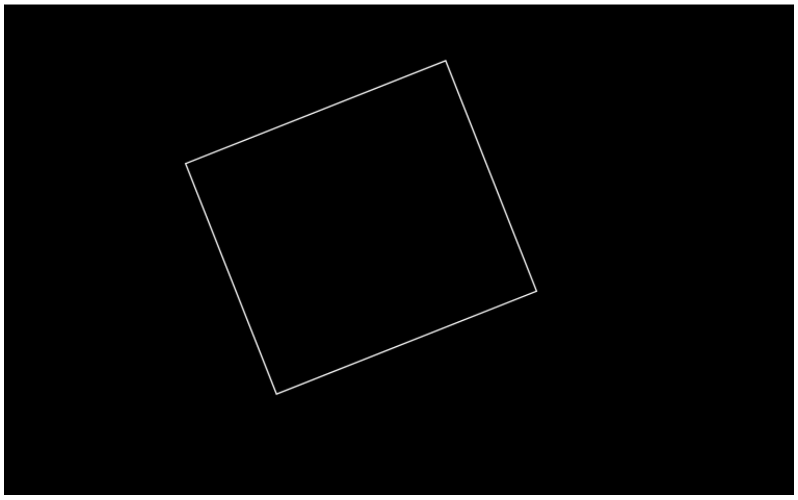
```
#include<iostream.h>
#include<conio.h>
#include<graphics.h>
#include<math.h>

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:



Program 14

Objective:
To draw Hermite Curve.

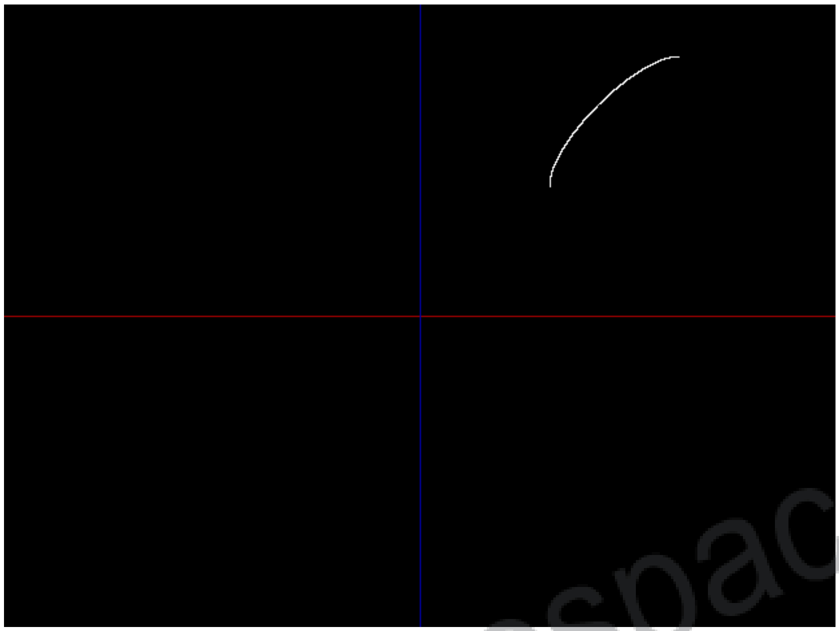
Code:

```
#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:



Program 15

Objective:

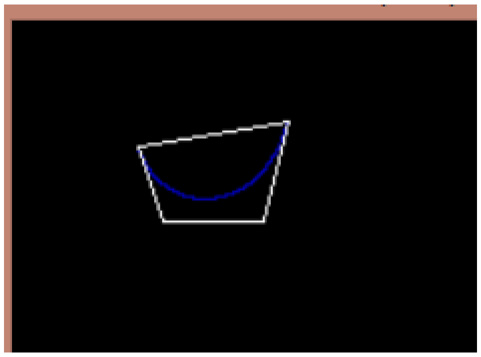
To draw Bezier Curve.

Code:

```
#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:



Program 16

Objective:

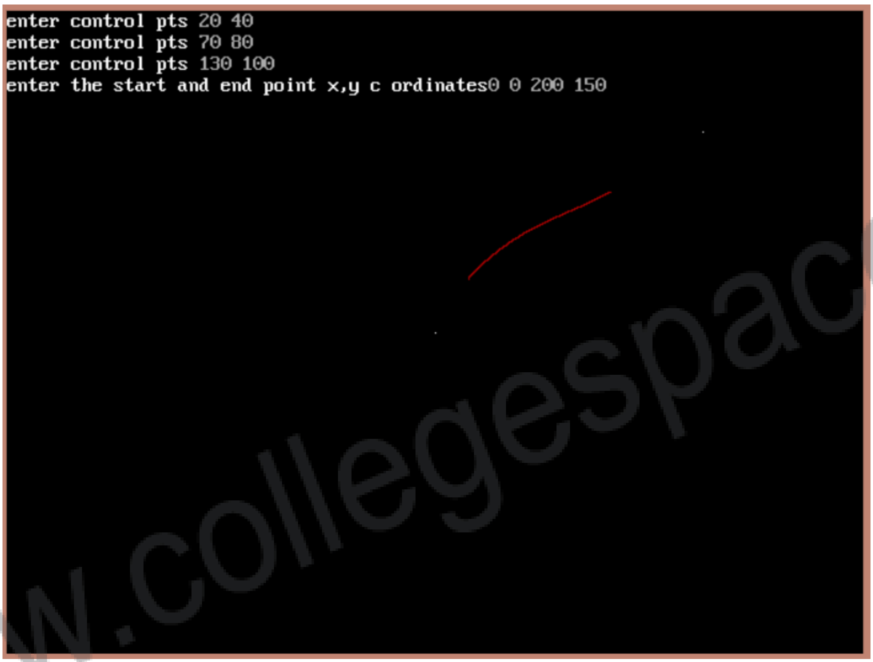
To draw B-Spline Curve.

Code:

```
#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 control pts 20 40
enter control pts 70 80
enter control pts 130 100
enter the start and end point x,y c ordinates0 0 200 150
```