**LAB FILE**

JAY KUMAR SINGH

2017UCO1691  COE-3 SEM-IV

**computer GRAPHICS CEC-12**



**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S.NO** | **PROGRAM** | **DATE** | **SIGN** |
| 1 | To draw line using the DDA approach |  |  |
| 2 | To draw line using the midpoint/bresenham approach |  |  |
| 3 | To draw circle using the midpoint/bresenham approach |  |  |
| 4 | To draw ellipse using the midpoint/bresenham approach |  |  |
| 5 | To draw parabola using the midpoint/bresenham approach |  |  |
| 6 | Rangoli rotating on a ellipse |  |  |
| 7 | To draw a rolling ball within a parabola |  |  |
| 8 | To draw a helicopter |  |  |
| 9 | To clip line using cyrus beck |  |  |
| 10 | To clip line using cohen sutherland |  |  |
| 11 | To clip line using midpoint subdivision |  |  |
| 12 | To clip line using nicholl lee nicholl |  |  |
| 13 | To clip line using liang barsky |  |  |
| 14 | To clip polygon using sutherland hodgeman |  |  |
| 15 | To print lists of vertices using weiler atherton |  |  |
| 16 | To display different views of a 3D figure |  |  |
| 17 | To display diff views of a 3D figure with hidden surface elimination |  |  |

PROGRAM – 1

**To draw line using the DDA approach**

#include<iostream>

#include<graphics.h>

using namespace std;

int abs (int n)

{

return ( (n>0) ? n : ( n \* (-1)));

}

void DDA(int X0, int Y0, int X1, int Y1)

{

int dx = X1 - X0;

int dy = Y1 - Y0;

int steps = abs(dx) > abs(dy) ? abs(dx) : abs(dy);

float Xinc = dx / (float) steps;

float Yinc = dy / (float) s teps;

float X = X0;

float Y = Y0;

for (int i = 0; i <= steps; i++)

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

delay(100);

}

}

int main()

{

int gd = DETECT, gm;

initgraph (&gd, &gm, "");

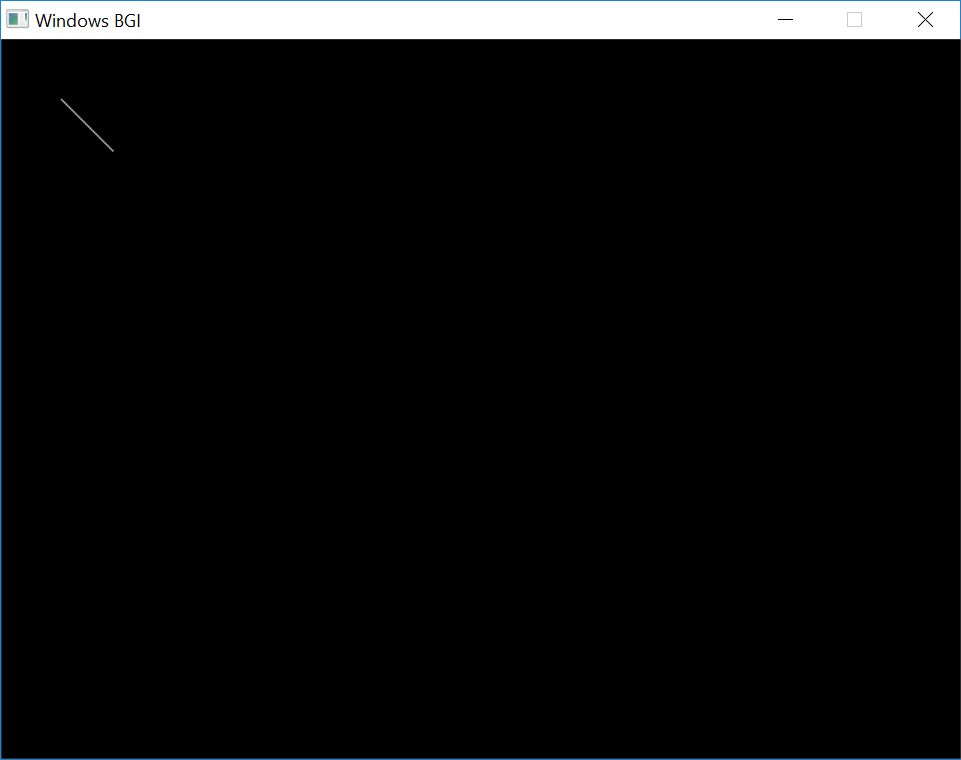
int x1,y1,x2,y2;

cin>>x1>>y1>>x2>>y2;

DDA(x1,y1,x2,y2);

return 0;

}



PROGRAM – 2

**To draw line using the midpoint/bresenham approach**

#include <iostream>

#include <graphics.h>

using namespace std;

void myline(float xa,float ya,float xb,float yb){

float x=xa;

float y=ya;

float dx=xb-xa;

float dy=yb-ya;

if(abs(dy)<abs(dx)){

float d=2\*dy-dx;

putpixel(x,y,WHITE);

while(x<=xb){

if(d<0){

d+=2\*dy;

}

else{

d+=2\*(dy-dx);

y++;

}

x++;

putpixel(x,y,WHITE);

}

}

else{

float d=2\*dx-dy;

putpixel(x,y,WHITE);

while(y<=yb){

if(d<0){

d+=dx;

}

else{

d+=2\*(dx-dy);

x++;

}

y++;

putpixel(x,y,WHITE);

}

}

}

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

int x1,y1,x2,y2;

cin>>x1>>y1>>x2>>y2;

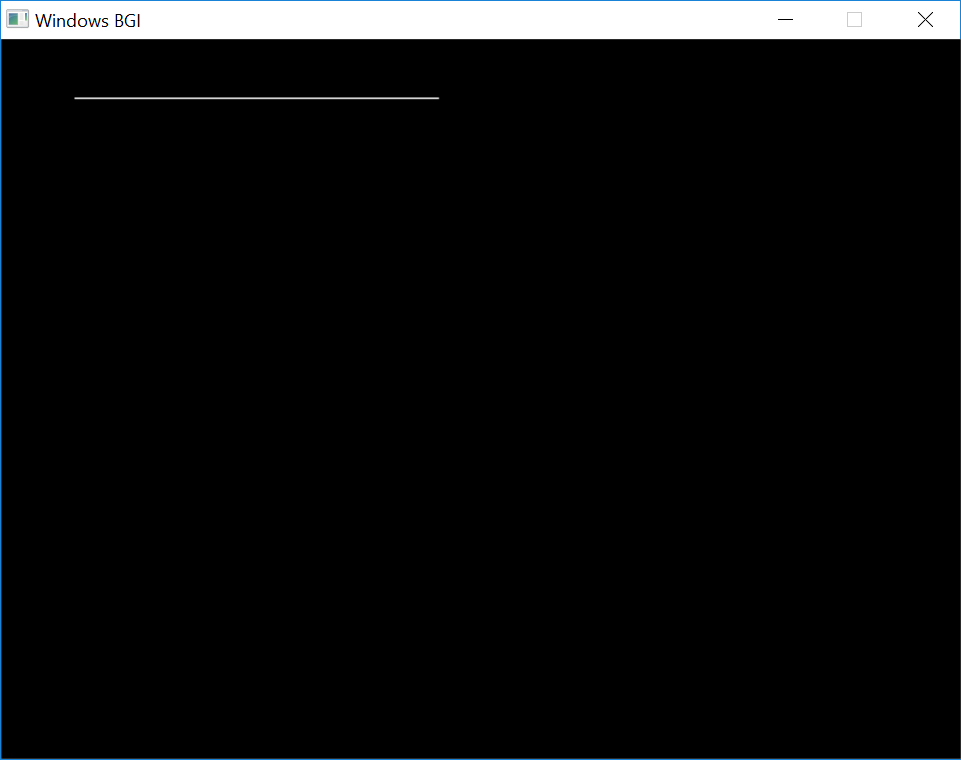
myline(x1,y1,x2,y2);

getch();

closegraph();

return 0;

}



PROGRAM – 3

**To draw circle using the midpoint/bresenham approach**

#include <iostream>

#include <graphics.h>

using namespace std;

void circlesymm(int x,int y,int a,int b){

putpixel(x+a,y+b,15);

putpixel(y+a,x+b,15);

putpixel(-x+a,y+b,15);

putpixel(x+a,-y+b,15);

putpixel(-x+a,-y+b,15);

putpixel(-y+a,-x+b,15);

putpixel(y+a,-x+b,15);

putpixel(-y+a,x+b,15);

}

void mycircle(int a,int b,int r){

int x=0,y=r,d=1-r;

circlesymm(x,y,a,b);

while(x<=y){

if(d<0)

d+=(2\*x+3);

else{

d+=(2\*(x-y)+5);

y--;

}

x++;

circlesymm(x,y,a,b);

}

}

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

int a,b,r;

cin>>a>>b>>r;

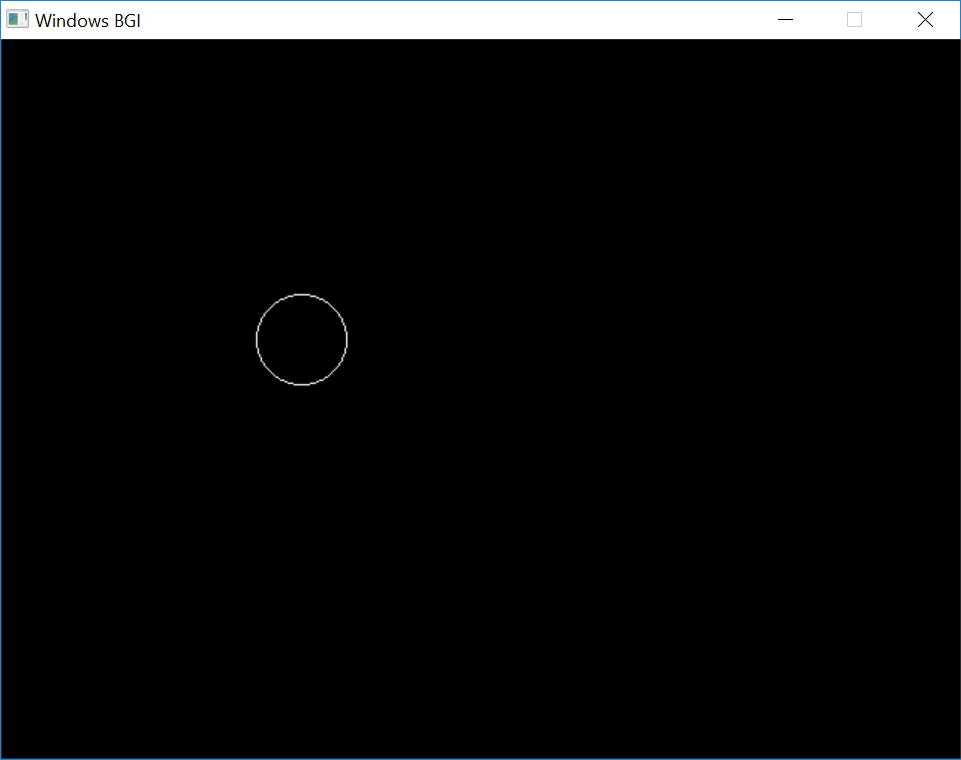
mycircle(a,b,r);

getch();

closegraph();

return 0;

}



PROGRAM – 4

**To draw ellipse using the midpoint/bresenham approach**

#include <iostream>

#include <graphics.h>

using namespace std;

void ellipsesymm(int x,int y,int c1,int c2){

putpixel(x+c1,y+c2,15);

putpixel(-x+c1,y+c2,15);

putpixel(x+c1,-y+c2,15);

putpixel(-x+c1,-y+c2,15);

}

void myellipse(int a,int b,int c1,int c2){

int x=a;

int y=0;

int d=(a\*a)-(1/4)\*(b\*b)-(a\*b\*b);

ellipsesymm(x,y,c1,c2);

while((b\*b\*x)>=(a\*a\*y)){

if(d<0)

d+=a\*a\*(3+(2\*y));

else{

d+=a\*a\*(2\*y+3)+(2\*b\*b\*(1-x));

x--;

}

y++;

ellipsesymm(x,y,c1,c2);

}

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

while(x>=0){

if(d<0){

d+=(b\*b\*(3-2\*x))+(2\*a\*a\*(1+y));

y++;

}

else

d+=(b\*b\*(3-2\*x));

x--;

ellipsesymm(x,y,c1,c2);

}

}

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

int a,b,c1,c2;

cin>>a>>b>>c1>>c2;

myellipse(a,b,c1,c2);

getch();

closegraph();

return 0;

}



PROGRAM – 5

**To draw parabola using the midpoint/bresenham approach**

#include <iostream>

#include <graphics.h>

using namespace std;

//y\*y=2\*p\*x

void parabolasymm(int x,int y,int a,int b){

putpixel(x+a,y+b,15);

putpixel(x+a,-y+b,15);

}

void myparabola(double x0,double y0,double p,double a,double b)

{

double x=x0;

double y=y0;

double d=(y0-0.5)\*(y0-0.5)-(2\*p\*(x0-1));

parabolasymm(x,y,a,b);

while(p<=y){

if(d>0){

d+=2\*(1-y+p);

y--;

}

else{

d+=2\*p;

}

x--;

parabolasymm(x,y,a,b);

}

d=(y-1)\*(y-1)-(2\*p\*(x-0.5));

while(y>0){

if(d<0){

d+=(3-2\*y+2\*p);

x--;

}

else

d+=(3-2\*y);

y--;

parabolasymm(x,y,a,b);

}

}

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

int x0,y0,p,a,b;

cin>>x0>>y0>>p>>a>>b;

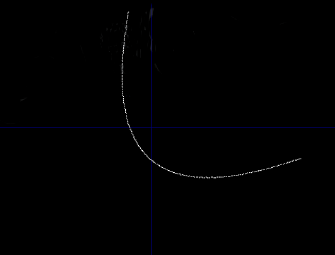
myparabola(x0,y0,p,a,b);

getch();

closegraph();

return 0;

}



PROGRAM – 6

**Rangoli rotating on ellipse**

#include <graphics.h>

#include <algorithm>

#include <bits/stdc++.h>

using namespace std;

void myArc2(int xc,int yc,int startangle,int stopangle,int radius,int theta,intomega,vector<pair<int,int> > &pts){

double x=radius\*cos((3.14159\*startangle)/180),y=radius\*sin((3.14159\*startangle)/180);

for(int X=startangle;X%360!=stopangle%360;X+=1){

x=xc+radius\*cos((3.14159\*X)/180);

y=yc-radius\*sin((3.14159\*X)/180);

int Tx=xc+(x-xc)\*cos(omega\*3.14/180)-(y-yc)\*sin(omega\*3.14/180);

int Ty=yc+(x-xc)\*sin(omega\*3.14/180)+(y-yc)\*cos(omega\*3.14/180);

pts.push\_back({Tx,Ty});

}

}

void rangoli(int Cx,int Cy,int radius,int theta=0,int omega=0,int color=WHITE){

int F=radius\*0.50,S=radius\*0.87;

pair <int,int> Cen[10]={{Cx+F,Cy+S},{Cx-F,Cy+S},{Cx-radius,Cy},{Cx-F,Cy-S},{Cx+F,Cy-S},{Cx+radius,Cy}};

for (int i=0;i<6;i++){

int temp=Cx+(Cen[i].first-Cx)\*cos(omega\*3.14/180)-(Cen[i].second-Cy)\*sin(omega\*3.14/180);

Cen[i].second=Cy+(Cen[i].first-Cx)\*sin(omega\*3.14/180)+(Cen[i].second-Cy)\*cos(omega\*3.14/180);

Cen[i].first=temp;

}

vector <pair <int,int> > pts;

myArc2(Cen[0].first,Cen[0].second,(+60),(+180),radius,theta,omega,pts);

myArc2(Cen[1].first,Cen[1].second,0,+120,radius,theta,omega,pts);

myArc2(Cen[2].first,Cen[2].second,+300,+60,radius,theta,omega,pts);

myArc2(Cen[3].first,Cen[3].second,240,+360,radius,theta,omega,pts);

myArc2(Cen[4].first,Cen[4].second,+180,+300,radius,theta,omega,pts);

myArc2(Cen[5].first,Cen[5].second,+120,+240,radius,theta,omega,pts);

srand(17);

random\_shuffle(pts.begin(),pts.end());///less flickering

circle(Cx,Cy,radius);

int COL=YELLOW;

for (auto x:pts)

putpixel(x.first,x.second,COL);

delay(1);

}

int main(){

int gd = DETECT, gm;

initwindow(1700,1700);

int a=120,b=80,C=350;

int radiusRangoli=30;

int theta=0;double omega=0;

int Cx=C,Cy=C;

int x0=Cx+a,y0=Cy;

while(1)

for (theta=0;theta<360;theta+=1){

ellipse(C,C,0,360,a,b);

x0=C+a\*cos(theta\*3.14/180);y0=C+b\*sin(theta\*3.14/180);

double R=sqrt((x0-C)(x0-C)+(y0-C)(y0-C));

double d\_omega=R/radiusRangoli;

omega+=d\_omega;

omega=fmod(omega,360);

Cx=x0+radiusRangoli\*cos(theta\*3.14/180);

Cy=y0+radiusRangoli\*sin(theta\*3.14/180);

rangoli(Cx,Cy,radiusRangoli,theta,omega);

clearviewport();

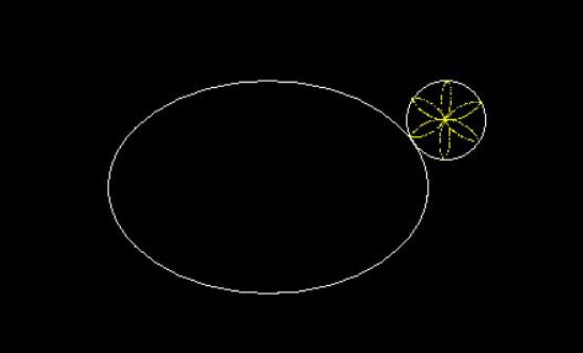
}

getch();

closegraph();

return 0;

}



PROGRAM – 7

**To draw a moving ball within a parabola**

#include <iostream>

#include <graphics.h>

using namespace std;

void circlesymm(int x,int y,int a,int b){

putpixel(x+a,y+b,15);

putpixel(y+a,x+b,15);

putpixel(-x+a,y+b,15);

putpixel(x+a,-y+b,15);

putpixel(-x+a,-y+b,15);

putpixel(-y+a,-x+b,15);

putpixel(y+a,-x+b,15);

putpixel(-y+a,x+b,15);

}

void mycircle(int a,int b,int r){

/\*a,b=centre r=radius\*/

int x=0;

int y=r;

int d=1-r;

circlesymm(x,y,a,b);

while(x<=y){

if(d<0)

d+=(2\*x+3);

else{

d+=(2\*(x-y)+5);

y--;

}

x++;

circlesymm(x,y,a,b);

}

}

void circlesymmb(int x,int y,int a,int b){

putpixel(x+a,y+b,0);

putpixel(y+a,x+b,0);

putpixel(-x+a,y+b,0);

putpixel(x+a,-y+b,0);

putpixel(-x+a,-y+b,0);

putpixel(-y+a,-x+b,0);

putpixel(y+a,-x+b,0);

putpixel(-y+a,x+b,0);

}

void mycircleb(int a,int b,int r){

/\*a,b=centre r=radius\*/

int x=0;

int y=r;

int d=1-r;

circlesymmb(x,y,a,b);

while(x<=y){

if(d<0)

d+=(2\*x+3);

else{

d+=(2\*(x-y)+5);

y--;

}

x++;

circlesymmb(x,y,a,b);

}

}

/\*

y\*y=2\*p\*x

\*/

void parabolasymm(int x,int y,int a,int b){

putpixel(x+a,y+b,15);

putpixel(x+a,-y+b,15);

}

void myparabola(double x0,double y0,double p,double a,double b)

{

double x=x0;

double y=y0;

double d=(y0-0.5)\*(y0-0.5)-(2\*p\*(x0-1));

parabolasymm(x,y,a,b);

while(p<=y){

if(d>0){

d+=2\*(1-y+p);

y--;

}

else{

d+=2\*p;

}

x--;

parabolasymm(x,y,a,b);

}

d=(y-1)\*(y-1)-(2\*p\*(x-0.5));

while(y>0){

if(d<0){

d+=(3-2\*y+2\*p);

x--;

}

else

d+=(3-2\*y);

y--;

parabolasymm(x,y,a,b);

}

}

void ball(double x0,double y0,double p,double a,double b,double r)

{

double x,y,d;

while(1){

x=x0;

y=y0;

d=(y0-0.5)\*(y0-0.5)-(2\*p\*(x0-1));

mycircle(x+a+r,y+b,r);

while(p<=y){

mycircleb(x+a+r,y+b,r);

myparabola(200,200,100,250,250);

if(d>0){

d+=2\*(1-y+p);

y--;

}

else{

d+=2\*p;

}

x--;

mycircle(x+a+r,y+b,r);

Sleep(50);

}

d=(y-1)\*(y-1)-(2\*p\*(x-0.5));

while(y>0){

mycircleb(x+a+r,y+b,r);

myparabola(200,200,100,250,250);

if(d<0){

d+=(3-2\*y+2\*p);

x--;

}

else

d+=(3-2\*y);

y--;

mycircle(x+a+r,y+b,r);

Sleep(50);

}

x=0;

y=0;

d=1-p;

mycircle(x+a+r,-y+b,r);

while(p>y){

mycircleb(x+a+r,-y+b,r);

myparabola(200,200,100,250,250);

if(d>0){

d+=((y+2)\*(y+2)-(2\*p\*(x+1.5)))-((y+1)\*(y+1)-(2\*p\*(x+0.5)));

x++;

}

else{

d+=((y+2)\*(y+2)-(2\*p\*(x+0.5)))-((y+1)\*(y+1)-(2\*p\*(x+0.5)));

}

y++;

mycircle(x+a+r,-y+b,r);

Sleep(50);

}

d=(y+0.5)\*(y+0.5)-(2\*p\*(x+1));

while(x<=x0){

mycircleb(x+a+r,-y+b,r);

myparabola(200,200,100,250,250);

if(d<0){

d+=((y+1.5)\*(y+1.5)-(2\*p\*(x+2)))-((y+0.5)\*(y+0.5)-(2\*p\*(x+1)));

y++;

}

else

d+=((y+0.5)\*(y+0.5)-(2\*p\*(x+2)))-((y+0.5)\*(y+0.5)-(2\*p\*(x+1)));

x++;

mycircle(x+a+r,-y+b,r);

Sleep(50);

}

mycircleb(x+a+r,-y+b,r);

x=x0;

y=y0;

d=(y0-0.5)\*(y0-0.5)-(2\*p\*(x0-1));

mycircle(x+a+r,-y+b,r);

while(p<=y){

mycircleb(x+a+r,-y+b,r);

myparabola(200,200,100,250,250);

if(d>0){

d+=2\*(1-y+p);

y--;

}

else{

d+=2\*p;

}

x--;

mycircle(x+a+r,-y+b,r);

Sleep(50);

}

d=(y-1)\*(y-1)-(2\*p\*(x-0.5));

while(y>0){

mycircleb(x+a+r,-y+b,r);

myparabola(200,200,100,250,250);

if(d<0){

d+=(3-2\*y+2\*p);

x--;

}

else

d+=(3-2\*y);

y--;

mycircle(x+a+r,-y+b,r);

Sleep(50);

}

mycircleb(x+a+r,-y+b,r);

x=0;

y=0;

d=1-p;

mycircle(x+a+r,y+b,r);

while(p>y){

mycircleb(x+a+r,y+b,r);

myparabola(200,200,100,250,250);

if(d>0){

d+=((y+2)\*(y+2)-(2\*p\*(x+1.5)))-((y+1)\*(y+1)-(2\*p\*(x+0.5)));

x++;

}

else{

d+=((y+2)\*(y+2)-(2\*p\*(x+0.5)))-((y+1)\*(y+1)-(2\*p\*(x+0.5)));

}

y++;

mycircle(x+a+r,y+b,r);

Sleep(50);

}

d=(y+0.5)\*(y+0.5)-(2\*p\*(x+1));

while(x<=x0){

mycircleb(x+a+r,y+b,r);

myparabola(200,200,100,250,250);

if(d<0){

d+=((y+1.5)\*(y+1.5)-(2\*p\*(x+2)))-((y+0.5)\*(y+0.5)-(2\*p\*(x+1)));

y++;

}

else

d+=((y+0.5)\*(y+0.5)-(2\*p\*(x+2)))-((y+0.5)\*(y+0.5)-(2\*p\*(x+1)));

x++;

mycircle(x+a+r,y+b,r);

Sleep(50);

}

mycircleb(x+a+r,y+b,r);

}

}

int main()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

myparabola(200,200,100,250,250);

ball(200,200,100,250,250,50);

getch();

closegraph();

return 0;

}

PROGRAM – 8

**To draw a helicopter**

#include <iostream>

#include <graphics.h>

#include <bits/stdc++.h>

#include <conio.h>

#define ScreenWidth getmaxx()

using namespace std;

#ifndef ELLIPSE\_H\_INCLUDED

#define ELLIPSE\_H\_INCLUDED

void myEllipse(int a, int b, int oX, int oY)

{

float d = a\*a + 0.25\*b\*b-a\*b\*b;

float x=a, y=0;

while(b\*b\*abs(x) > a\*a\*abs(y))

{

if(d<0)

{

d+=a\*a\*(3+2\*y);

}

else

{

d+=a\*a\*(2\*y+3)+2\*b\*b\*(1-x);

x--;

}

y++;

putpixel(x+oX, y+oY, WHITE);

putpixel(-x+oX, y+oY, WHITE);

putpixel(-x+oX, -y+oY, WHITE);

putpixel(x+oX, -y+oY, WHITE);

}

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

putpixel(x+oX, y+oY, WHITE);

putpixel(-x+oX, y+oY, WHITE);

putpixel(-x+oX, -y+oY, WHITE);

putpixel(x+oX, -y+oY, WHITE);

while(x>0)

{

if(d<0)

{

d+=b\*b\*(3-2\*x)+a\*a\*(2\*y+2);

y++;

}

else{

d+=b\*b\*(3-2\*x);

}

x--;

putpixel(x+oX, y+oY, WHITE);

putpixel(-x+oX, y+oY, WHITE);

putpixel(-x+oX, -y+oY, WHITE);

putpixel(x+oX, -y+oY, WHITE);

}

}

#endif // LINES\_H\_INCLUDED

int main()

{

cout << "Hello world!" << endl;

int gd=DETECT,gm,x=0;

//Change BGI directory according to yours

char driver[]="C:\\TURBOC3\\BGI";

initgraph(&gd,&gm,driver);

// ellipse(100, 60,100,100,100,100);

int colors[8]= {1,2,3,4,14,10,12,7};

int i=0;

int originX=100, originY=100;

myEllipse(100,50, originX, originY);

myEllipse(25, 10, originX, originY);

/\* float c = sqrt(a\*a\* + b\*b);

float x1 = -a\*a\*m/c;

float y1 = b\*b/c;

\*/

// float c = sqrt(200\*200+100\*100);

//float c1 = sqrt();

while(!kbhit()){

cleardevice();

myEllipse(100,50, originX+1, originY);

myEllipse(25, 10, originX+1, originY);

float Xdiag = 50, xdiag = 12.5;

int A = 100, B= 50;

int a = 25, b = 10;

float Xver = originX;

float xver = originX;

float Yver = 50;

float yver = 10;

float Yhor = originY;

float yhor = originY;

float Xhor = 100;

float xhor = 25;

float ydiag = sqrt(a\*a\*b\*b - b\*b\*xdiag\*xdiag)/a;

float Ydiag = sqrt(A\*A\*B\*B - B\*B\*Xdiag\*Xdiag)/A;

setcolor(colors[i%8]);

line(Xver, Yver+originY, xver, yver+originY);

setcolor(colors[(i+1)%8]);

line(Xver, -Yver+originY, xver, -yver+originY);

setcolor(colors[(i+2)%8]);

line(Xhor+originX, Yhor, xhor+originX, yhor);

setcolor(colors[(i+3)%8]);

line(-Xhor+originX, Yhor, -xhor+originX, yhor);

setcolor(colors[(i+4)%8]);

line(xdiag+originX, ydiag+originY, Xdiag+originX, Ydiag+originY);

setcolor(colors[(i+5)%8]);

line(-xdiag+originX, ydiag+originY, -Xdiag+originX, Ydiag+originY);

setcolor(colors[(i+6)%8]);

line(-xdiag+originX, -ydiag+originY, -Xdiag+originX, -Ydiag+originY);

setcolor(colors[(i+7)%8]);

line(xdiag+originX, -ydiag+originY, Xdiag+originX, -Ydiag+originY);

originX=(originX+2)%ScreenWidth;

i++;

setcolor(WHITE);

line(originX, originY+B, originX, originY+50+B);

line(originX+140, originY+B+50, originX-140, originY+B+50);

line(originX+140, originY+B+50, originX+140, originY+B+150);

line(originX-140, originY+B+50, originX-140, originY+B+150);

line(originX+140, originY+B+150, originX-140, originY+B+150);

rectangle(originX+90+20, originY+B+50+20, originX+80+50, originY+B+50+70);

myEllipse(20,20,originX+100, originY+B+170);

myEllipse(20,20,originX-100, originY+B+170);

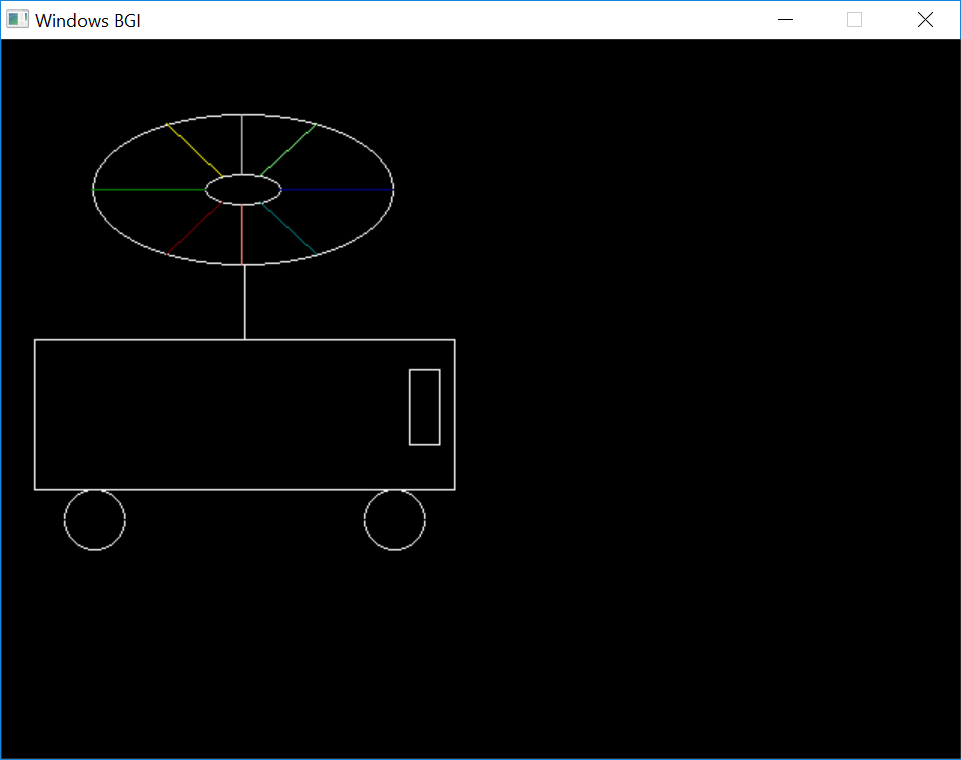
Sleep(200);

}

getch();

return 0;

}



PROGRAM – 9

**To clip line using cyrus beck**

#include <iostream>

#include <vector>

#include <graphics.h>

#include <algorithm>

using namespace std;

struct pt{

float x,y;

};

pt diff(pt a,pt b){

pt temp;

temp.x=b.x-a.x;

temp.y=b.y-a.y;

return temp;

}

pt normal(pt ab){

pt temp=ab;

float c=temp.x;

temp.x=temp.y;

temp.y=c;

if(temp.x>0&&temp.y>0)

temp.x\*=-1;

else if(temp.x<0&&temp.y>0)

temp.y\*=-1;

else if(temp.x<0&&temp.y<0)

temp.x\*=-1;

else

temp.y\*=-1;

return temp;

}

float dotproduct(pt ab,pt cd){

float res;

res=ab.x\*cd.x+ab.y\*cd.y;

return res;

}

pt sum (pt a, pt b){

pt temp;

temp.x = a.x + b.x;

temp.y = a.y +b.y;

return temp;

}

pt mulScalar(float t, pt ab) {

ab.x\*=t;

ab.y\*=t;

return ab;

}

vector <pair < pt , float > > cyrusbeck(pt p1,pt p2,pt\* cw, int len){

vector <pair < pt , float > > res;

vector <pair<float, float> > tees;

for (int i =0; i<len; i++){

pt a=cw[i];

pt b=cw[(i+1)%len];

pt ab = diff(a,b);

pt Nab = normal(ab);

pt ap1 = diff(a,p1);

float dot1 = dotproduct(Nab, ap1);

float dot2 = -1\* dotproduct(Nab, diff(p1,p2));

float t = float((float)dot1/(float)dot2);

if (t >0 && t<1){

tees.push\_back(make\_pair(t, dot2>0?0:1));

}

}

sort(tees.begin(), tees.end());

for (float i =0; i<tees.size(); i++){

pt temp = sum(p1,mulScalar(tees[i].first, diff(p1,p2)));

res.push\_back(make\_pair(temp, tees[i].second));

}

return res;

}

int main()

{

int n;

pt p1,p2;

cout<<"Enter the vertices of the line\n";

cin>>p1.x>>p1.y>>p2.x>>p2.y;

cout<<"Enter the size of the clipping window\n";

cin>>n;

cout<<"Enter the vertices of the clipping window\n";

pt\* cw=new pt[n];

for(int i=0;i<n;i++)

cin>>cw[i].x>>cw[i].y;

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

line(p1.x,p1.y,p2.x,p2.y);

for(int i=0;i<n;i++)

line(cw[i].x,cw[i].y,cw[(i+1)%n].x,cw[(i+1)%n].y);

delay(2000);

vector<pair<pt,float> > res=cyrusbeck(p1,p2,cw,n);

if(res.size()==2){

line(res[0].first.x,res[0].first.y,res[1].first.x,res[1].first.y);

setcolor(BLACK);

line(p1.x,p1.y,res[0].first.x,res[0].first.y);

line(res[1].first.x,res[1].first.y,p2.x,p2.y);

}

else if(res.size()==1){

if(res[0].second==1){

line(p1.x,p1.y,res[0].first.x,res[0].first.y);

setcolor(BLACK);

line(res[0].first.x,res[0].first.y,p2.x,p2.y);

}

else{

line(res[0].first.x,res[0].first.y,p2.x,p2.y);

setcolor(BLACK);

line(p1.x,p1.y,res[0].first.x,res[0].first.y);

}

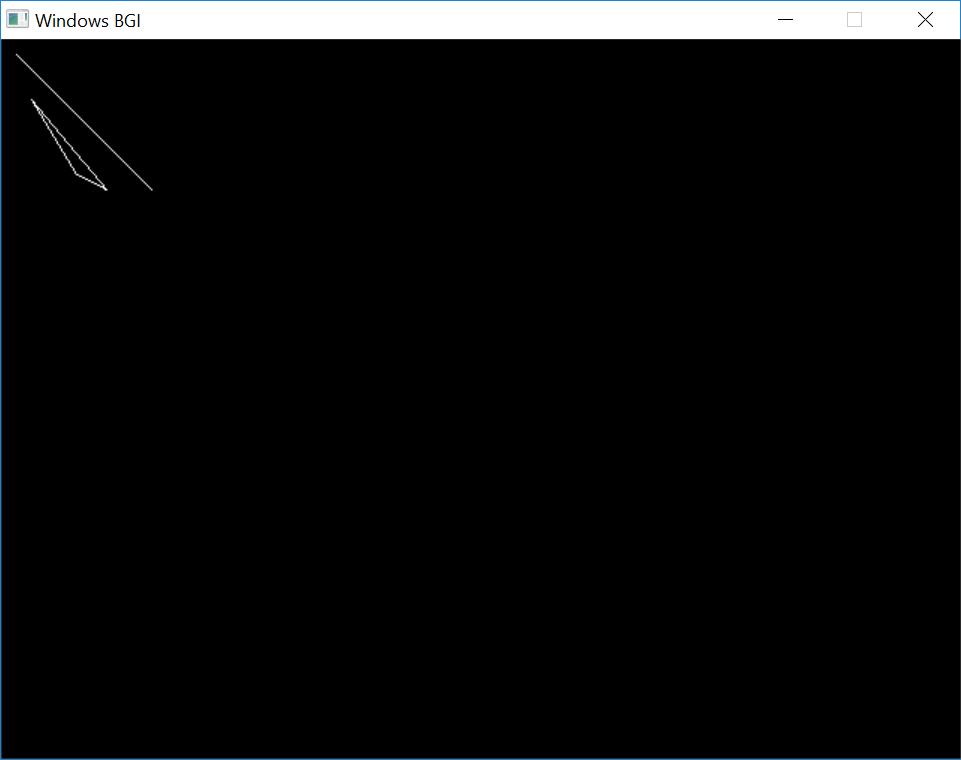
}

getch();

closegraph();

return 0;

}



PROGRAM – 10

**To clip line using cohen sutherland**

#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(YELLOW);

line(xwmin,ywmin,xwmin,ywmax);

line(xwmin,ywmin,xwmax,ywmin);

line(xwmax,ywmin,xwmax,ywmax);

line(xwmin,ywmax,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()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

float xwmin, xwmax, ywmin, ywmax;

cout << "Enter the x limits for the clipping window : ";

cin >> xwmin >> xwmax;

cout <<"\n Enter the y limits for 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 : ";

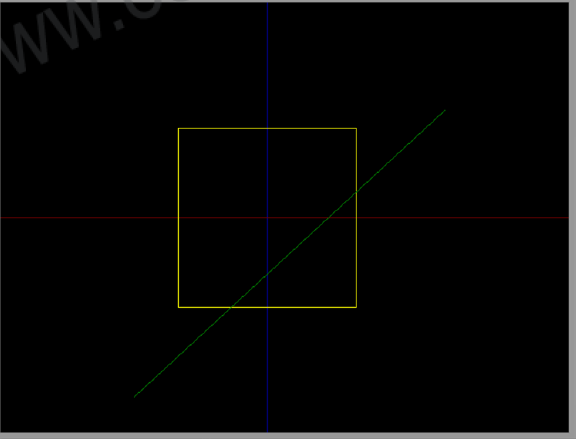
cin>>x2>>y2;

clipLine(x1+300,240-y1,x2+300,240-y2,xwmin+300,240-ywmin,xwmax+300,240-ywmax);

getch();

closegraph();

}



PROGRAM – 11

**To clip line using midpoint subdivision**

#include<iostream>

#include<graphics.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()

{

int gd=DETECT,gm;

initgraph(&gd,&gm,"C:\\TC\\BGI");

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);

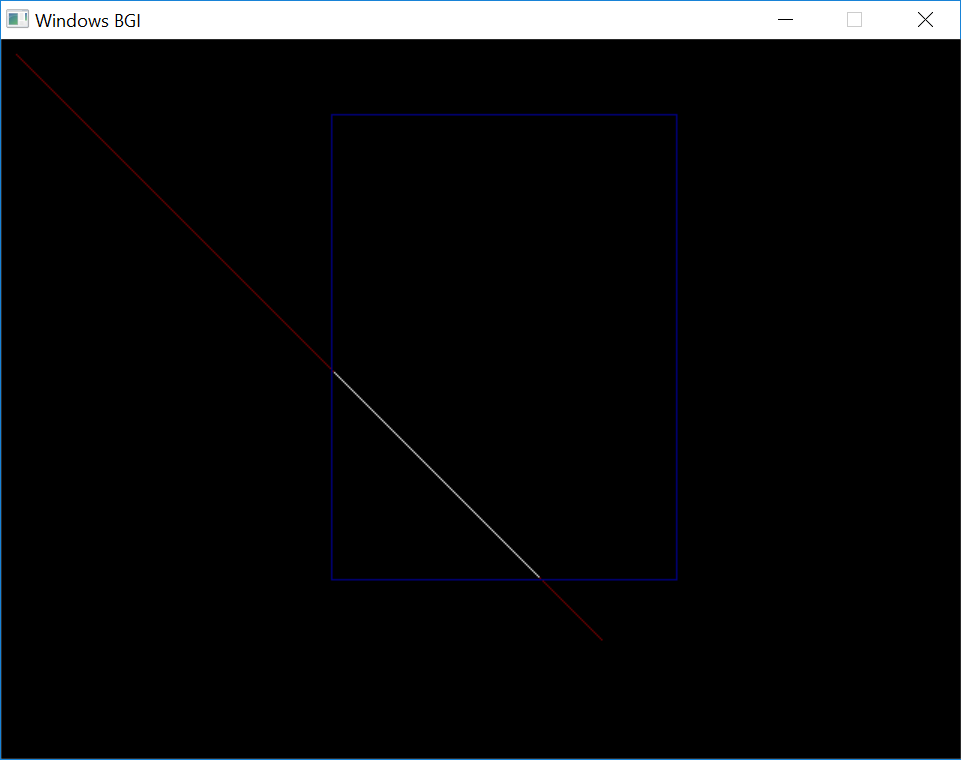
clipLineMid(winMin,winMax,p1,p2);

getch();

closegraph();

return 0;

}



PROGRAM – 12

**To clip line using nicholl lee nicholl**

# include <conio.h>

# include <graphics.h>

# include <math.h>

# include <iostream>

using namespace std;

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);

int main()

{

int x1,y1,x2,y2;

int gdriver = DETECT, gmode;

int ch;

float m;

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);

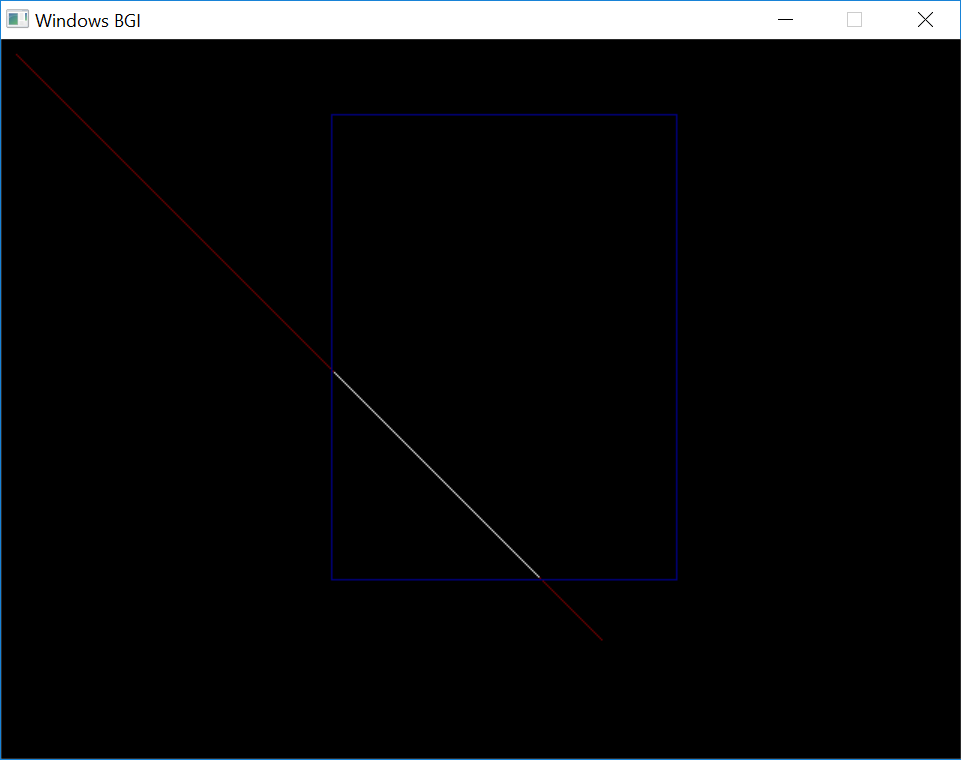
line(a+x1,b-y1,a+xmin,b-ymax);

setcolor(5);

line(a+nx1,b-ny1,a+nx2,b-ny2);

}

}



PROGRAM – 13

**To clip line using liang barsky**

#include<iostream>

#include<graphics.h>

#include<math.h>

#include<dos.h>

using namespace std;

int main()

{

int i,gd=DETECT,gm;

int x1,y1,x2,y2,xmin,xmax,ymin,ymax,xx1,xx2,yy1,yy2,dx,dy;

float t1,t2,p[4],q[4],temp;

x1=120;

y1=120;

x2=300;

y2=300;

initgraph(&gd,&gm,"c:\\turboc3\\bgi");

setcolor(RED);

line(x1,y1,x2,y2);

setcolor(WHITE);

xmin=100;

ymin=100;

xmax=250;

ymax=250;

rectangle(xmin,ymin,xmax,ymax);

delay(1000);

dx=x2-x1;

dy=y2-y1;

p[0]=-dx;

p[1]=dx;

p[2]=-dy;

p[3]=dy;

q[0]=x1-xmin;

q[1]=xmax-x1;

q[2]=y1-ymin;

q[3]=ymax-y1;

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

{

if(p[i]==0)

{

cout<<"line is parallel to one of the clipping boundary";

if(q[i]>=0)

{

if(i<2)

{

if(y1<ymin)

{

y1=ymin;

}

if(y2>ymax)

{

y2=ymax;

}

line(x1,y1,x2,y2);

}

if(i>1)

{

if(x1<xmin)

{

x1=xmin;

}

if(x2>xmax)

{

x2=xmax;

}

line(x1,y1,x2,y2);

}

}

}

}

t1=0;

t2=1;

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

{

temp=q[i]/p[i];

if(p[i]<0)

{

if(t1<=temp)

t1=temp;

}

else

{

if(t2>temp)

t2=temp;

}

}

if(t1<t2)

{

xx1 = x1 + t1 \* p[1];

xx2 = x1 + t2 \* p[1];

yy1 = y1 + t1 \* p[3];

yy2 = y1 + t2 \* p[3];

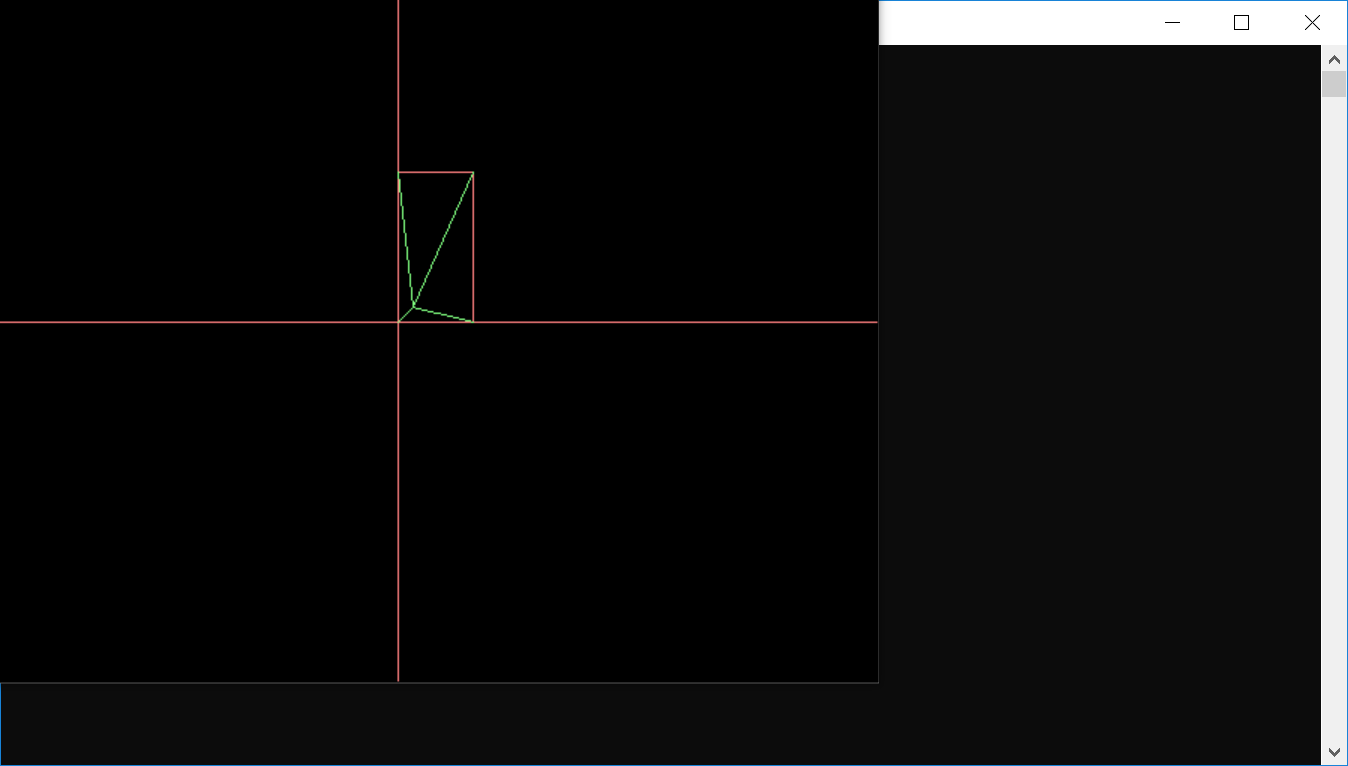
line(xx1,yy1,xx2,yy2);

}

delay(5000);

closegraph();

}



PROGRAM – 14

**To clip polygon using sutherland hodgeman**

// C++ program for implementing Sutherland–Hodgman

// algorithm for polygon clipping

#include<iostream>

#include <graphics.h>

using namespace std;

const int MAX\_POINTS = 20;

// Returns x-value of point of intersectipn of two

// lines

int x\_intersect(int x1, int y1, int x2, int y2,

int x3, int y3, int x4, int y4)

{

int num = (x1\*y2 - y1\*x2) \* (x3-x4) -

(x1-x2) \* (x3\*y4 - y3\*x4);

int den = (x1-x2) \* (y3-y4) - (y1-y2) \* (x3-x4);

return num/den;

}

// Returns y-value of point of intersectipn of

// two lines

int y\_intersect(int x1, int y1, int x2, int y2,

int x3, int y3, int x4, int y4)

{

int num = (x1\*y2 - y1\*x2) \* (y3-y4) -

(y1-y2) \* (x3\*y4 - y3\*x4);

int den = (x1-x2) \* (y3-y4) - (y1-y2) \* (x3-x4);

return num/den;

}

// This functions clips all the edges w.r.t one clip

// edge of clipping area

void clip(int poly\_points[][2], int &poly\_size,

int x1, int y1, int x2, int y2)

{

int new\_points[MAX\_POINTS][2], new\_poly\_size = 0;

// (ix,iy),(kx,ky) are the co-ordinate values of

// the points

for (int i = 0; i < poly\_size; i++)

{

// i and k form a line in polygon

int k = (i+1) % poly\_size;

int ix = poly\_points[i][0], iy = poly\_points[i][1];

int kx = poly\_points[k][0], ky = poly\_points[k][1];

// Calculating position of first point

// w.r.t. clipper line

int i\_pos = (x2-x1) \* (iy-y1) - (y2-y1) \* (ix-x1);

// Calculating position of second point

// w.r.t. clipper line

int k\_pos = (x2-x1) \* (ky-y1) - (y2-y1) \* (kx-x1);

// Case 1 : When both points are inside

if (i\_pos < 0 && k\_pos < 0)

{

//Only second point is added

new\_points[new\_poly\_size][0] = kx;

new\_points[new\_poly\_size][1] = ky;

new\_poly\_size++;

}

// Case 2: When only first point is outside

else if (i\_pos >= 0 && k\_pos < 0)

{

// Point of intersection with edge

// and the second point is added

new\_points[new\_poly\_size][0] = x\_intersect(x1,

y1, x2, y2, ix, iy, kx, ky);

new\_points[new\_poly\_size][1] = y\_intersect(x1,

y1, x2, y2, ix, iy, kx, ky);

new\_poly\_size++;

new\_points[new\_poly\_size][0] = kx;

new\_points[new\_poly\_size][1] = ky;

new\_poly\_size++;

}

// Case 3: When only second point is outside

else if (i\_pos < 0 && k\_pos >= 0)

{

//Only point of intersection with edge is added

new\_points[new\_poly\_size][0] = x\_intersect(x1,

y1, x2, y2, ix, iy, kx, ky);

new\_points[new\_poly\_size][1] = y\_intersect(x1,

y1, x2, y2, ix, iy, kx, ky);

new\_poly\_size++;

}

// Case 4: When both points are outside

else

{

//No points are added

}

}

// Copying new points into original array

// and changing the no. of vertices

poly\_size = new\_poly\_size;

for (int i = 0; i < poly\_size; i++)

{

poly\_points[i][0] = new\_points[i][0];

poly\_points[i][1] = new\_points[i][1];

}

}

// Implements Sutherland–Hodgman algorithm

void suthHodgClip(int poly\_points[][2], int poly\_size,

int clipper\_points[][2], int clipper\_size)

{

//i and k are two consecutive indexes

for (int i=0; i<clipper\_size; i++)

{

int k = (i+1) % clipper\_size;

// We pass the current array of vertices, it's size

// and the end points of the selected clipper line

clip(poly\_points, poly\_size, clipper\_points[i][0],

clipper\_points[i][1], clipper\_points[k][0],

clipper\_points[k][1]);

}

// Printing vertices of clipped polygon

setcolor(RED);

for (int i=0; i < poly\_size; i++){

line(poly\_points[i][0],poly\_points[i][1],poly\_points[(i+1)%poly\_size][0],poly\_points[(i+1)%poly\_size][1]);

}

}

int main()

{

int gd=DETECT;

int gm;

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

// Defining polygon vertices in clockwise order

int poly\_size = 3;

int poly\_points[20][2] = {{100,150}, {200,250},

{300,200}};

// Defining clipper polygon vertices in clockwise order

// 1st Example with square clipper

int clipper\_size = 4;

int clipper\_points[][2] = {{150,150}, {150,200},

{200,200}, {200,150} };

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

line(clipper\_points[i][0],clipper\_points[i][1],clipper\_points[(i+1)%4][0],clipper\_points[(i+1)%4][1]);

}

delay(3000);

setcolor(YELLOW);

for(int i=0;i<3;i++){

line(poly\_points[i][0],poly\_points[i][1],poly\_points[(i+1)%3][0],poly\_points[(i+1)%3][1]);

}

delay(3000);

// 2nd Example with triangle clipper

/\*int clipper\_size = 3;

int clipper\_points[][2] = {{100,300}, {300,300},

{200,100}};\*/

//Calling the clipping function

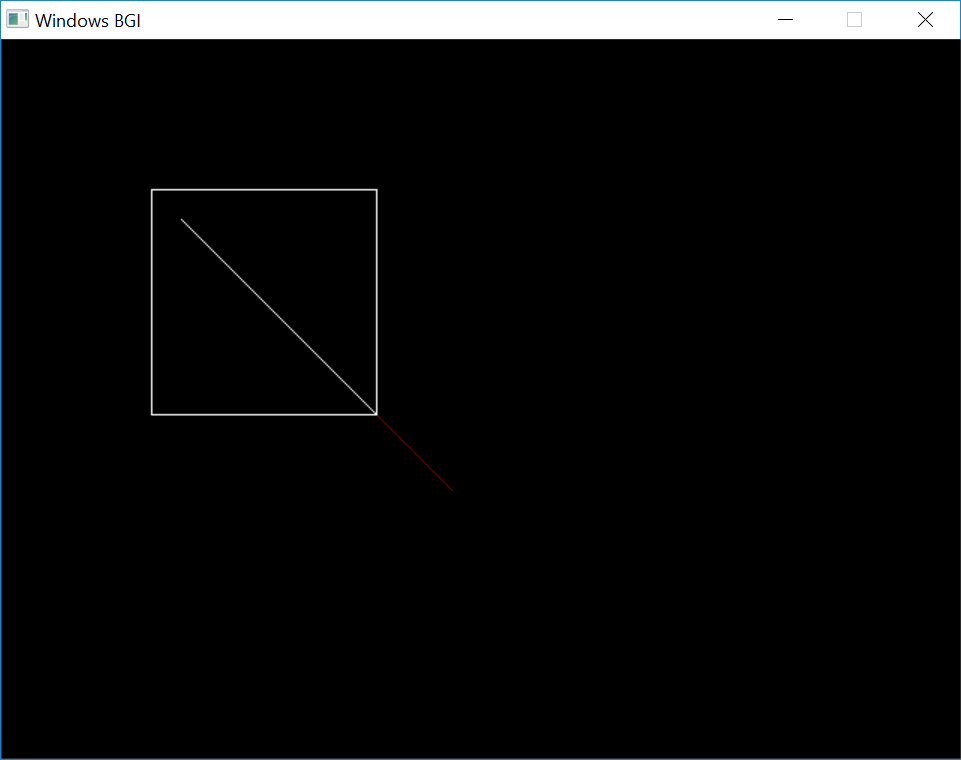
suthHodgClip(poly\_points, poly\_size, clipper\_points,

clipper\_size);

getch();

return 0;

}



PROGRAM – 15

**To print lists of vertices using weiler atherton**

#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct pt{

float x,y;

};

pt diff(pt a,pt b){

pt temp;

temp.x=b.x-a.x;

temp.y=b.y-a.y;

return temp;

}

pt normal(pt ab){

pt temp=ab;

float c=temp.x;

temp.x=temp.y;

temp.y=c;

if(temp.x>0&&temp.y>0)

temp.x\*=-1;

else if(temp.x<0&&temp.y>0)

temp.y\*=-1;

else if(temp.x<0&&temp.y<0)

temp.x\*=-1;

else

temp.y\*=-1;

return temp;

}

float dotproduct(pt ab,pt cd){

float res;

res=ab.x\*cd.x+ab.y\*cd.y;

return res;

}

pt sum (pt a, pt b){

pt temp;

temp.x = a.x + b.x;

temp.y = a.y +b.y;

return temp;

}

pt mulScalar(float t, pt ab) {

ab.x\*=t;

ab.y\*=t;

return ab;

}

vector <pair < pt , float > > cyrusbeck(pt p1,pt p2,pt\* cw, int len){

vector <pair < pt , float > > res;

vector <pair<float, float> > tees;

for (int i =0; i<len; i++){

pt a=cw[i];

pt b=cw[(i+1)%len];

pt ab = diff(a,b);

pt Nab = normal(ab);

pt ap1 = diff(a,p1);

float dot1 = dotproduct(Nab, ap1);

float dot2 = -1\* dotproduct(Nab, diff(p1,p2));

float t = float((float)dot1/(float)dot2);

if (t >0 && t<1){

tees.push\_back(make\_pair(t, dot2>0?0:1));

}

}

sort(tees.begin(), tees.end());

for (float i =0; i<tees.size(); i++){

pt temp = sum(p1,mulScalar(tees[i].first, diff(p1,p2)));

res.push\_back(make\_pair(temp, tees[i].second));

}

return res;

}

int main()

{

int m,n;

cout<<"Enter the size of the subject polygon\n";

cin>>m;

cout<<"Enter the vertices of the subject polygon\n";

pt\* sp=new pt[m];

for(int i=0;i<m;i++)

cin>>sp[i].x>>sp[i].y;

cout<<"Enter the size of the clipping window\n";

cin>>n;

cout<<"Enter the vertices of the clipping window\n";

pt\* cw=new pt[n];

for(int i=0;i<n;i++)

cin>>cw[i].x>>cw[i].y;

vector <pair < pt , float > > nsp;

vector <pair < pt , float > > ncw;

for(int i=0;i<m;i++){

pt p1=sp[i];

pt p2=sp[(i+1)%m];

nsp.push\_back(make\_pair(p1,-1));

vector<pair<pt,float> > res=cyrusbeck(p1,p2,cw,n);

for(float i=0;i<res.size();i++)

nsp.push\_back(res[i]);

}

for(int i=0;i<n;i++){

pt p1=cw[i];

pt p2=cw[(i+1)%n];

ncw.push\_back(make\_pair(p1,-1));

vector<pair<pt,float> >res=cyrusbeck(p1,p2,sp,m);

for(float i=0;i<res.size();i++)

ncw.push\_back(res[i]);

}

cout<<'\n';

for(float i=0;i<nsp.size();i++)

cout<<nsp[i].first.x<<" "<<nsp[i].first.y<<" "<<nsp[i].second<<'\n';

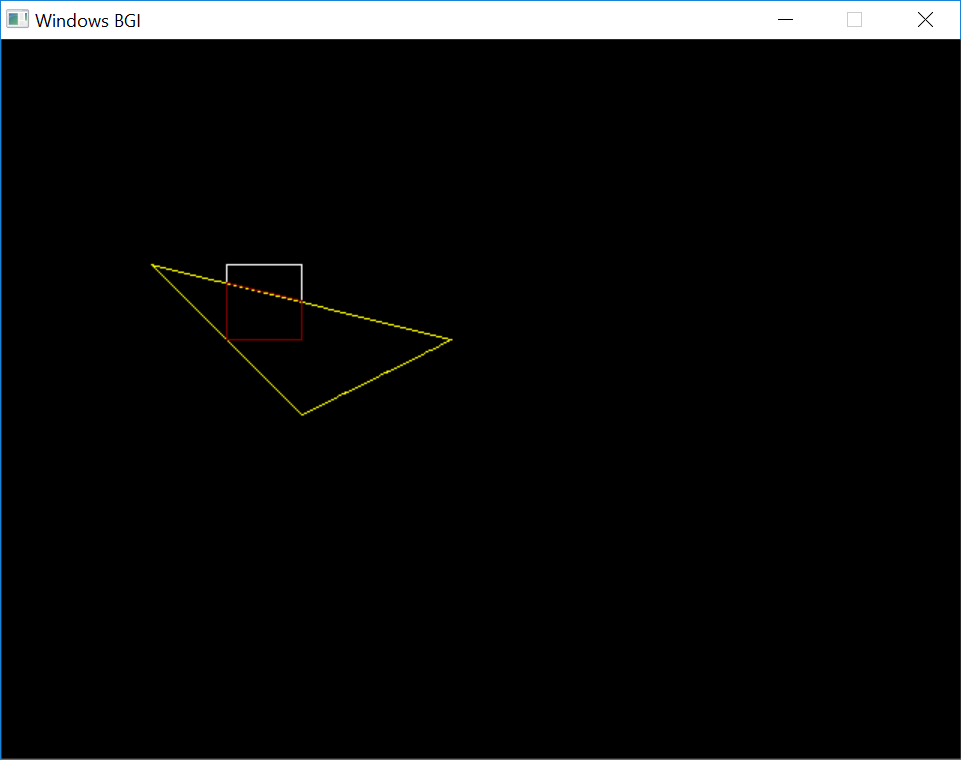
cout<<'\n';

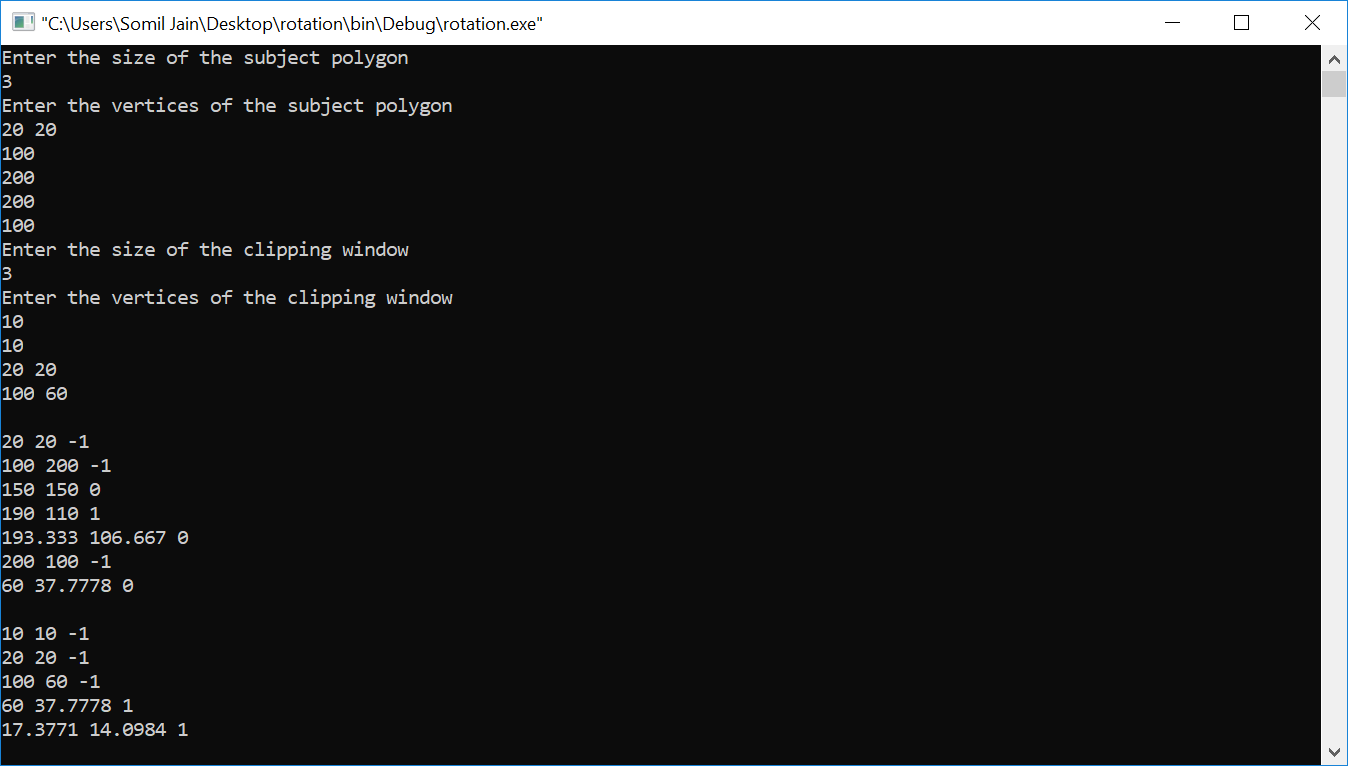
for(float i=0;i<ncw.size();i++)

cout<<ncw[i].first.x<<" "<<ncw[i].first.y<<" "<<ncw[i].second<<'\n';

return 0;

}





PROGRAM – 16

**To display different views of a 3D figure**

#include <bits/stdc++.h>

#include <graphics.h>

using namespace std;

double pi =3.14159265359;

struct point{

double x,y,z,w;

};

vector<point> Multiply\_matrices(double rotation\_mat[4][4],vector<point> verts){

int i,j;

vector<point> NewVertices(verts.size());

for(j=0;j<verts.size();j++){

NewVertices[j].x= verts[j].x\*rotation\_mat[0][0]+ verts[j].y\*rotation\_mat[0][1]+verts[j].z\*rotation\_mat[0][2]+verts[j].w\*rotation\_mat[0][3];

NewVertices[j].y= verts[j].x\*rotation\_mat[1][0]+ verts[j].y\*rotation\_mat[1][1]+verts[j].z\*rotation\_mat[1][2]+verts[j].w\*rotation\_mat[1][3];

NewVertices[j].z= verts[j].x\*rotation\_mat[2][0]+ verts[j].y\*rotation\_mat[2][1]+verts[j].z\*rotation\_mat[2][2]+verts[j].w\*rotation\_mat[2][3];

NewVertices[j].w= verts[j].x\*rotation\_mat[3][0]+ verts[j].y\*rotation\_mat[3][1]+verts[j].z\*rotation\_mat[3][2]+verts[j].w\*rotation\_mat[3][3];

}

return NewVertices;

}

struct Object{

private:

vector<point> Vertices;

/\*vector<point> Vertices; \*/

vector<pair<int,int> > Edges;

public:

void initialise\_vertices(){

int n;

cin>>n;

point A;

for(int i=0;i<n;i++){

cin>>A.x>>A.y>>A.z;

A.w=1;

Vertices.push\_back(A);

}

cin>>n;

pair<int,int> edge;

for(int i=0;i<n;i++){

cin>>edge.first>>edge.second;

Edges.push\_back(edge);

}

}

void Rotate\_about\_x(double theta){

double rotation\_mat[4][4]={1,0,0,0,0,cos(theta),-sin(theta),0,0,sin(theta),cos(theta),0,0,0,0,1};

Vertices=Multiply\_matrices(rotation\_mat,Vertices);

}

void Rotate\_about\_y(double theta){

double rotation\_mat[4][4]={cos(theta),0,sin(theta),0,0,1,0,0,-sin(theta),0,cos(theta),0,0,0,0,1};

Vertices=Multiply\_matrices(rotation\_mat,Vertices);

}

void Rotate\_about\_z(double theta){

double rotation\_mat[4][4]={cos(theta),-sin(theta),0,0,sin(theta),cos(theta),0,0,0,0,1,0,0,0,0,1};

Vertices=Multiply\_matrices(rotation\_mat,Vertices);

}

void front\_view(int color=YELLOW){//Viewing along Z axis

int xc= getmaxx()/6;

int yc=getmaxy()/4;

int i;

for(i=0;i<Edges.size();i++){

setcolor(color);

line(Vertices[Edges[i].first].x+xc,yc-Vertices[Edges[i].first].y,Vertices[Edges[i].second].x+xc,yc-Vertices[Edges[i].second].y);

}

}

void side\_view(int color=YELLOW){//Viewing along X axis

int xc= getmaxx()/2;

int yc=getmaxy()/4;

int i;

for(i=0;i<Edges.size();i++){

setcolor(color);

line(-Vertices[Edges[i].first].z+xc,yc-Vertices[Edges[i].first].y,-Vertices[Edges[i].second].z+xc,yc-Vertices[Edges[i].second].y);

}

}

void top\_view(int color=YELLOW){//Viewing along Y axis

int xc= (5\*getmaxx() )/6;

int yc=getmaxy()/4;

int i;

for(i=0;i<Edges.size();i++){

setcolor(color);

line(Vertices[Edges[i].first].x+xc,yc- (-Vertices[Edges[i].first].z),Vertices[Edges[i].second].x+xc,yc-(-Vertices[Edges[i].second].z));

}

}

void isometric\_view(int color=YELLOW)

{

int xc=getmaxx()/6;

int yc=(3\*getmaxy())/4;

vector<point>temp=Vertices;

Rotate\_about\_y(pi/4);

Rotate\_about\_x(atan( (double) 1 / sqrt( (double) 2)));

for(int i=0;i<Edges.size();i++){

setcolor(color);

line(Vertices[Edges[i].first].x+xc,yc-Vertices[Edges[i].first].y,Vertices[Edges[i].second].x+xc,yc-Vertices[Edges[i].second].y);

}

Vertices=temp;

}

void dimetric\_view(int color=YELLOW)

{

int xc=getmaxx()/2;

int yc=(3\*getmaxy() )/4;

vector<point>temp=Vertices;

double k=1.3;

Rotate\_about\_y(asin( k/sqrt((double) 2)));

Rotate\_about\_x(asin(k/sqrt( (double) (k\*k)+ 2)));

for(int i=0;i<Edges.size();i++){

setcolor(color);

line(Vertices[Edges[i].first].x+xc,yc-Vertices[Edges[i].first].y,Vertices[Edges[i].second].x+xc,yc-Vertices[Edges[i].second].y);

}

Vertices=temp;

}

};

void divide\_screen(int color=YELLOW){

int x=getmaxx();

int y=getmaxy();

int first\_thirdx=x/3,second\_thirdx=(2\*x)/3;

setcolor(color);

line(first\_thirdx,0,first\_thirdx,y);

setcolor(color);

line(second\_thirdx,0,second\_thirdx,y);

setcolor(color);

line(0,y/2,x,y/2);

}

int main(){

int gd = DETECT,gm;

initgraph(&gd , &gm,"C:\\TC\\BGI");

divide\_screen();

Object prism;

bool x\_rotation=false,y\_rotation=false,z\_rotation=false;

prism.initialise\_vertices();

int x\_angle=5,y\_angle=5,z\_angle=5;

char ch;

double radian;

while(true){

prism.front\_view(YELLOW);

prism.top\_view(YELLOW);

prism.side\_view(YELLOW);

prism.isometric\_view(YELLOW);

prism.dimetric\_view(YELLOW);

ch=getch();

if(ch=='S'||ch=='s')

break;

if(ch=='X'||ch=='x'){

prism.front\_view(BLACK);

prism.top\_view(BLACK);

prism.side\_view(BLACK);

prism.isometric\_view(BLACK);

prism.dimetric\_view(BLACK);

prism.Rotate\_about\_x(( x\_angle\* pi )/180 );

}

if(ch=='Y'||ch=='y'){

prism.front\_view(BLACK);

prism.top\_view(BLACK);

prism.side\_view(BLACK);

prism.isometric\_view(BLACK);

prism.dimetric\_view(BLACK);

prism.Rotate\_about\_y(( y\_angle\* pi )/180 );

}

if(ch=='Z'||ch=='z'){

prism.front\_view(BLACK);

prism.top\_view(BLACK);

prism.side\_view(BLACK);

prism.isometric\_view(BLACK);

prism.dimetric\_view(BLACK);

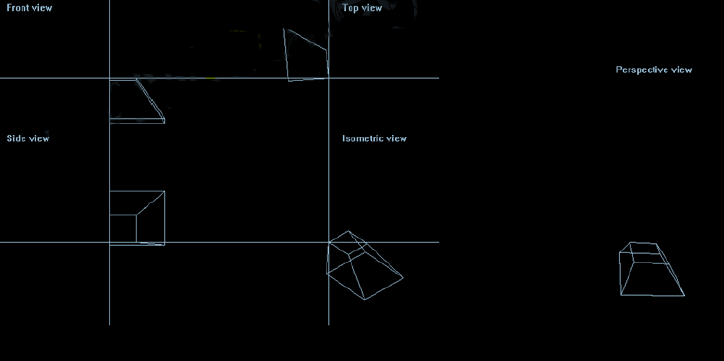
prism.Rotate\_about\_z(( z\_angle\* pi )/180 );

}

}

return 0;

}



PROGRAM – 17

To display different views of a 3D figure with hidden surface elimination