**PROGRAM 1**

**DDA LINE DRAWING ALGORITHM**

#include<iostream>

#include<graphics.h>

using namespace std;

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

int main()

{

initwindow(1500,1500);

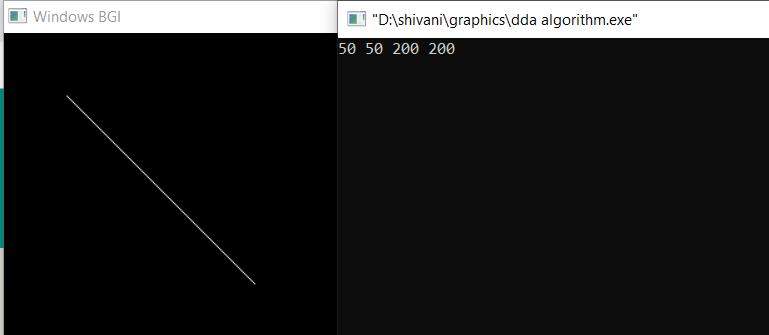
int x1,y1,x2,y2;

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

dda(x1,y1,x2,y2);

getch();

}



**PROGRAM 2**

**BRESENHAM LINE DRAWING ALGORITHM**

#include<bits/stdc++.h>

#include<graphics.h>

using namespace std;

void Bresenham(float x1,float y1,float x2,float y2)

{

float x,y;

float dx=x2-x1;

float dy=y2-y1;

x=x1;

y=y1;

float p=2\*dy-dx;

while(x<x2)

{

if(p<0)

{

putpixel(x,y,WHITE);

p=p+2\*dy;

}

else

{

putpixel(x,y,WHITE);

p=p+2\*dy-2\*dx;

y=y+1;

}

x++;

}

}

int main()

{

initwindow(1000,1000);

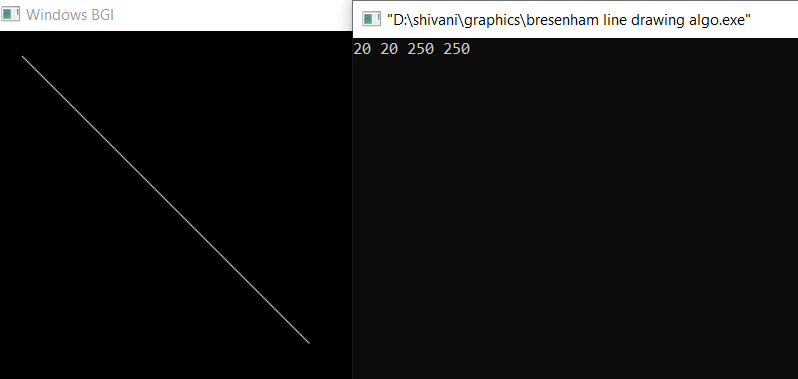
float x1,x2,y1,y2;

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

Bresenham(x1,y1,x2,y2);

getch();

}



**PROGRAM 3**

**BRESENHAM CIRCLE DRAWING ALGORITHM**

#include<iostream>

#include<graphics.h>

using namespace std;

void bresenham\_circle(int r,int xc,int yc)

{

int x=0;

int y=r;

int d=3-2\*r;

while(y>=x)

{

x++;

if(d<0)

{

d=d+4\*x+6;

}

else if(d>=0)

{

d=d+4\*x-4\*y+10;

y=y-1;

}

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc-y,WHITE);

putpixel(xc+y,yc+x,WHITE);

putpixel(xc-y,yc+x,WHITE);

putpixel(xc+y,yc-x,WHITE);

putpixel(xc-y,yc-x,WHITE);

}

}

int main()

{

initwindow(1000,1000);

int gm,gd=DETECT;

int r,x,y;

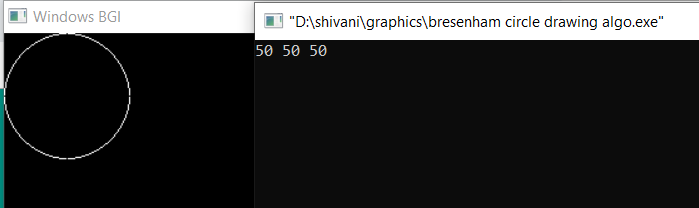
cin>>r>>x>>y;

bresenham\_circle(r,x,y);

getch();

return 0;

}



**PROGRAM 4**

**MIDPOINT CIRCLE DRAWING ALGORITHM**

#include<iostream>

#include<graphics.h>

using namespace std;

void midpoint\_circle(int r,int xc,int yc)

{

int x=0,y=r;

int p=1-r;

putpixel(x,y,WHITE);

while(x<y)

{

if(p<0)

{

p=p+2\*x+1;

x+=1;

}

else

{

p=p+2\*x-2\*y+1;

x+=1;

y--;

}

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc-y,WHITE);

putpixel(xc+y,yc+x,WHITE);

putpixel(xc-y,yc+x,WHITE);

putpixel(xc+y,yc-x,WHITE);

putpixel(xc-y,yc-x,WHITE);

}

}

int main()

{

initwindow(1000,1000);

int gm,gd=DETECT;

int r,x,y;

cout<<"Enter the radius of the circle: "<<endl;

cin>>r;

cout<<"Enter the coordinates of center: "<<endl;

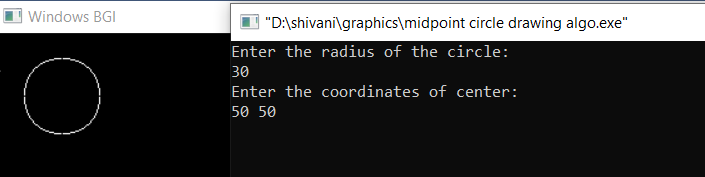
cin>>x>>y;

midpoint\_circle(r,x,y);

getch();

return 0;

}



**PROGRAM 5**

**MIDPOINT ELLIPSE DRAWING ALGORITHM**

#include<graphics.h>

#include<iostream>

using namespace std;

void ellipse(int rx,int ry,int xc,int yc)

{

float dx,dy,d1,d2,x,y;

x=0;

y=ry;

d1=(ry\*ry)-(rx\*rx\*ry)+(0.25\*rx\*rx); // Initial decision parameter of region 1

dx=2\*ry\*ry\*x;

dy=2\*rx\*rx\*y;

while(dx<dy) // For region 1

{

// Print points based on 4-way symmetry

putpixel(x+xc,y+yc,WHITE);

putpixel(-x+xc,y+yc,WHITE);

putpixel(x+xc,-y+yc,WHITE);

putpixel(-x+xc,-y+yc,WHITE);

// Checking and updating value of

// decision parameter based on algorithm

if(d1<0)

{

x++;

dx=dx+(2\*ry\*ry);

d1=d1+dx+(ry\*ry);

}

else

{

x++;

y--;

dx=dx+(2\*ry\*ry);

dy=dy-(2\*rx\*rx);

d1=d1+dx-dy+(ry\*ry);

}

}

// Decision parameter of region 2

d2=((ry\*ry)\*((x+0.5)\*(x+0.5)))+((rx\*rx)\*((y-1)\*(y-1)))-(rx\*rx\*ry\*ry);

// Plotting points of region 2

while (y >= 0)

{

// Print points based on 4-way symmetry

putpixel(x+xc,y+yc,WHITE);

putpixel(-x+xc,y+yc,WHITE);

putpixel(x+xc,-y+yc,WHITE);

putpixel(-x+xc,-y+yc,WHITE);

// Checking and updating parameter

// value based on algorithm

if(d2>0)

{

y--;

dy=dy-(2\*rx\*rx);

d2=d2+(rx\*rx)-dy;

}

else

{

y--;

x++;

dx=dx+(2\*ry\*ry);

dy=dy-(2\*rx\*rx);

d2=d2+dx-dy+(rx\*rx);

}

}

}

int main()

{

int xc,yc,rx,ry;

initwindow(1000,1000);

cout<<"Enter the center coordinates of ellipse: "<<endl;

cin>>xc>>yc;

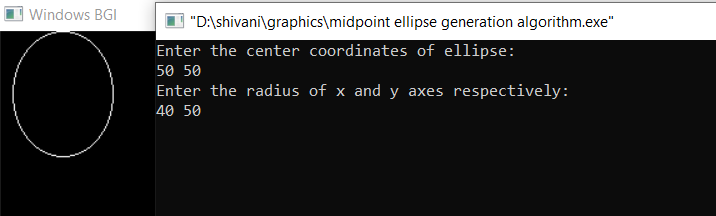
cout<<"Enter the radius of x and y axes respectively: "<<endl;

cin>>rx>>ry;

ellipse(rx,ry,xc,yc);

getch();

}



**PROGRAM 6**

**ROTATE A POINT ABOUT ORIGIN**

#include <math.h>

#include<iostream>

#include <graphics.h>

#define PI 22/7

using namespace std;

int main()

{

int gd=DETECT,gm;

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

float x,y,theta;

cout<<"Enter the point to be rotated: "<<endl;

cin>>x>>y;

cout<<"Enter the angle: "<<endl;

cin>>theta;

float angle = (PI\*theta)/180;

cout<<"("<<x<<","<<y<<")"<<endl;

putpixel(x,y,WHITE);

float m=(cos(angle)\*x)+(-(sin(angle))\*y);

float n=(sin(angle)\*x)+(cos(angle)\*y);

line(x,y,m,n);

cout<<"Point after rotation: "<<endl;

cout<<m<<","<<n<<endl;

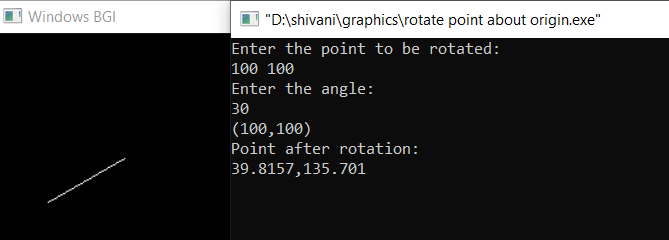
putpixel(m,n,WHITE);

getch();

closegraph();

return 0;

}



**PROGRAM 7**

**ROTATE A TRIANGLE ABOUT ORIGIN**

#include<iostream>

#include<graphics.h>

#include<math.h>

#define PI 22/7

using namespace std;

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

void triangle(float x1,float y1,float x2,float y2,float x3,float y3,float angle)

{

float a[3][3];

float p[3][3];

float ans[3][3];

a[0][0] = cos(angle); //anticlockwise rotation

a[1][0] = sin(angle);

a[0][1] = -(sin(angle));

a[1][1] = cos(angle);

a[0][2] = 0;

a[1][2] = 0;

a[2][0] = 0;

a[2][1] = 0;

a[2][2] = 1;

p[0][0]=x1;

p[1][0]=y1;

p[2][0]=1;

p[0][1]=x2;

p[1][1]=y2;

p[2][1]=1;

p[0][2]=x3;

p[1][2]=y3;

p[2][2]=1;

int i,j,k;

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

{

for(j = 0; j < 3; j++)

ans[i][j]=0;

}

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

for(j = 0; j < 3; j++){

for(k = 0; k < 3; k++)

ans[i][j] += a[i][k] \* p[k][j];

}

}

dda(ans[0][0],ans[1][0],ans[0][1],ans[1][1]);

dda(ans[0][0],ans[1][0],ans[0][2],ans[1][2]);

dda(ans[0][2],ans[1][2],ans[0][1],ans[1][1]);

delay(5000);

}

int main()

{

int gd=DETECT,gm;

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

float x1,y1,x2,y2,x3,y3,theta;

cout<<"Enter coordinates of triangle:"<<endl;

cin>>x1>>y1>>x2>>y2>>x3>>y3;

cout<<"enter the angle:"<<endl;

cin>>theta;

float angle = (PI\*theta)/180;

dda(x1,y1,x2,y2);

dda(x2,y2,x3,y3);

dda(x1,y1,x3,y3);

triangle(x1,y1,x2,y2,x3,y3,angle);

/\* a[0][0] = cos(angle); //clockwise rotation

a[1][0] = sin(angle);

a[0][1] = sin(angle);

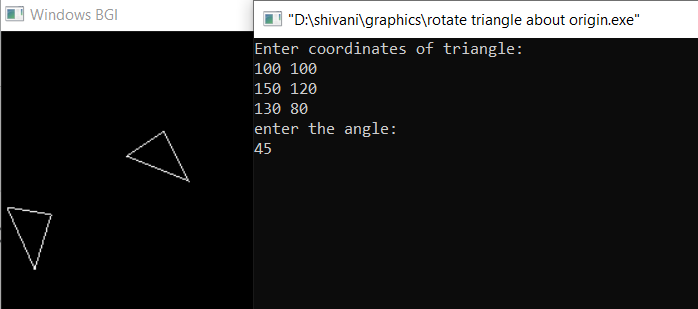
a[1][1] = cos(angle); \*/

getch();

closegraph();

return 0;

}



**PROGRAM 8**

**SCALE A TRIANGLE ABOUT ORIGIN**

#include<iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

int main()

{

int gd=DETECT, gm;

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

int x1,y1,x2,y2,x3,y3,X1,Y1,X2,Y2,X3,Y3,sx,sy;

cout<<"enter the triangle coordinates"<<endl;

cin>>x1>>y1>>x2>>y2>>x3>>y3;

cout<<"enter the value of scaling in x-axis"<<endl;

cin>>sx;

cout<<"enter the value of scaling in y-axis"<<endl;

cin>>sy;

dda(x1,y1,x2,y2);

dda(x2,y2,x3,y3);

dda(x1,y1,x3,y3);

X1=sx\*x1;

Y1=y1\*sy;

X2=x2\*sx;

Y2=y2\*sy;

X3=x3\*sx;

Y3=y3\*sy;

dda(X1,Y1,X2,Y2);

dda(X2,Y2,X3,Y3);

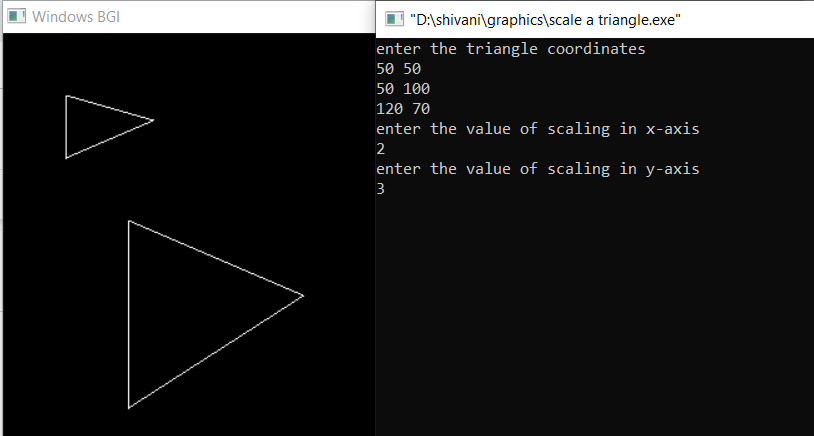
dda(X1,Y1,X3,Y3);

getch();

closegraph();

return 0;

}



**PROGRAM 9**

**TRANSLATE A TRIANGLE ABOUT ORIGIN**

#include<iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

int main()

{

int gd=DETECT, gm;

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

int x1,y1,x2,y2,x3,y3,X1,Y1,X2,Y2,X3,Y3,tx,ty;

cout<<"enter the triangle coordinates"<<endl;

cin>>x1>>y1>>x2>>y2>>x3>>y3;

cout<<"enter the value of scaling in x-axis"<<endl;

cin>>tx;

cout<<"enter the value of scaling in y-axis"<<endl;

cin>>ty;

dda(x1,y1,x2,y2);

dda(x2,y2,x3,y3);

dda(x1,y1,x3,y3);

X1=x1+tx;

Y1=y1+ty;

X2=x2+tx;

Y2=y2+ty;

X3=x3+tx;

Y3=y3+ty;

dda(X1,Y1,X2,Y2);

dda(X2,Y2,X3,Y3);

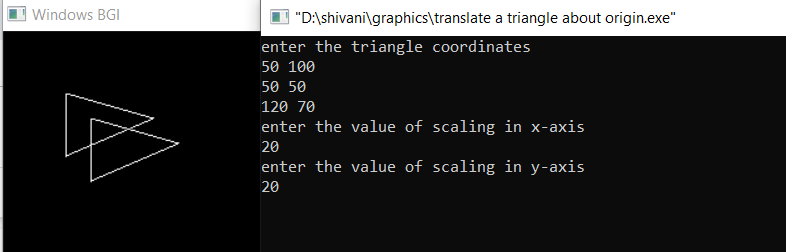
dda(X1,Y1,X3,Y3);

getch();

closegraph();

return 0;

}



**PROGRAM 10**

**REFLECT A TRIANGLE ABOUT ORIGIN**

#include<iostream>

#include<graphics.h>

#include<math.h>

using namespace std;

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

int main()

{

int gd=DETECT, gm;

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

int x1,y1,x2,y2,x3,y3,X1,Y1,X2,Y2,X3,Y3,m,b;

cout<<"enter the triangle coordinates"<<endl;

cin>>x1>>y1>>x2>>y2>>x3>>y3;

cout<<"enter the value of slope"<<endl;

cin>>m;

cout<<"enter the value of constant"<<endl;

cin>>b;

dda(x1,y1,x2,y2);

dda(x2,y2,x3,y3);

dda(x1,y1,x3,y3);

X1=(x1\*(1-m\*m))/(1+m\*m)+ (2\*m\*y1)/(1+m\*m)-(2\*b\*m)/(1+m\*m);

Y1=(2\*x1\*m)/(1+m\*m)+(y1\*(m\*m-1))/(1+m\*m)+(2\*b)/(1+m\*m);

X2=(x2\*(1-m\*m))/(1+m\*m)+ (2\*m\*y2)/(1+m\*m)-(2\*b\*m)/(1+m\*m);

Y2=(2\*x2\*m)/(1+m\*m)+(y2\*(m\*m-1))/(1+m\*m)+(2\*b)/(1+m\*m);

X3=(x3\*(1-m\*m))/(1+m\*m)+ (2\*m\*y3)/(1+m\*m)-(2\*b\*m)/(1+m\*m);

Y3=(2\*x3\*m)/(1+m\*m)+(y3\*(m\*m-1))/(1+m\*m)+(2\*b)/(1+m\*m);

dda(X1,Y1,X2,Y2);

dda(X2,Y2,X3,Y3);

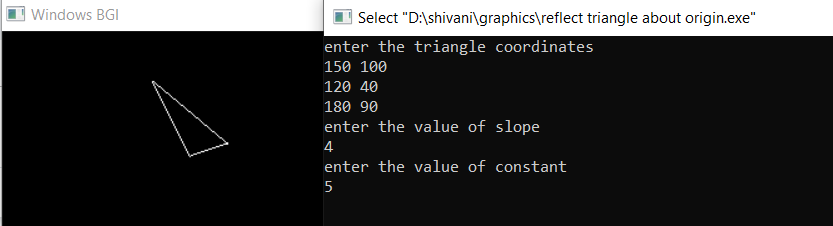
dda(X3,Y3,X1,Y1);

getch();

closegraph();

return 0;

}



**PROGRAM 11**

**COHEN SUTHERLAND LINE CLIPPING ALFORITHM**

#include <iostream>

#include <graphics.h>

using namespace std;

const int INSIDE = 0; // 0000

const int LEFT = 1; // 0001

const int RIGHT = 2; // 0010

const int BOTTOM = 4; // 0100

const int TOP = 8; // 1000

const int x\_max = 200;

const int y\_max = 200;

const int x\_min = 40;

const int y\_min = 40;

int computeCode(double x, double y)

{

// initialized as being inside

int code = INSIDE;

if (x < x\_min) // to the left of rectangle

code |= LEFT;

else if (x > x\_max) // to the right of rectangle

code |= RIGHT;

if (y < y\_min) // below the rectangle

code |= BOTTOM;

else if (y > y\_max) // above the rectangle

code |= TOP;

return code;

}

// Implementing Cohen-Sutherland algorithm

// Clipping a line from P1 = (x2, y2) to P2 = (x2, y2)

void cohenSutherlandClip(double x1, double y1,

double x2, double y2)

{

// Compute region codes for P1, P2

int code1 = computeCode(x1, y1);

int code2 = computeCode(x2, y2);

// Initialize line as outside the rectangular window

bool accept = false;

while (true)

{

if ((code1 == 0) && (code2 == 0))

{

// If both endpoints lie within rectangle

accept = true;

break;

}

else if (code1 & code2)

{

// If both endpoints are outside rectangle,

// in same region

break;

}

else

{

// Some segment of line lies within the

// rectangle

int code\_out;

double x, y;

// At least one endpoint is outside the

// rectangle, pick it.

if (code1 != 0)

code\_out = code1;

else

code\_out = code2;

// Find intersection point;

// using formulas y = y1 + slope \* (x - x1),

// x = x1 + (1 / slope) \* (y - y1)

if (code\_out & TOP)

{

// point is above the clip rectangle

x = x1 + (x2 - x1) \* (y\_max - y1) / (y2 - y1);

y = y\_max;

}

else if (code\_out & BOTTOM)

{

// point is below the rectangle

x = x1 + (x2 - x1) \* (y\_min - y1) / (y2 - y1);

y = y\_min;

}

else if (code\_out & RIGHT)

{

// point is to the right of rectangle

y = y1 + (y2 - y1) \* (x\_max - x1) / (x2 - x1);

x = x\_max;

}

else if (code\_out & LEFT)

{

// point is to the left of rectangle

y = y1 + (y2 - y1) \* (x\_min - x1) / (x2 - x1);

x = x\_min;

}

// Now intersection point x,y is found

// We replace point outside rectangle

// by intersection point

if (code\_out == code1)

{

x1 = x;

y1 = y;

code1 = computeCode(x1, y1);

}

else

{

x2 = x;

y2 = y;

code2 = computeCode(x2, y2);

}

}

}

if (accept)

{

cout <<"Line accepted from " << x1 << ", "

<< y1 << " to "<< x2 << ", " << y2 << endl;

line(x1,y1,x2,y2);

getch();

// Here the user can add code to display the rectangle

// along with the accepted (portion of) lines

}

else

cout << "Line rejected" << endl;

}

// Driver code

int main()

{

int gd = DETECT, gm;

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

line(x\_min,y\_min,x\_max,y\_min);

line(x\_min,y\_min,x\_min,y\_max);

line(x\_min,y\_max,x\_max,y\_max);

line(x\_max,y\_min,x\_max,y\_max);

int x1,y1,x2,y2;

cout<<"enter 1st Line coordinates"<<endl;

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

cohenSutherlandClip(x1,y1,x2,y2);

cout<<"enter 2nd Line coordinates"<<endl;

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

cohenSutherlandClip(x1,y1,x2,y2);

cout<<"enter 3rd Line coordinates"<<endl;

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

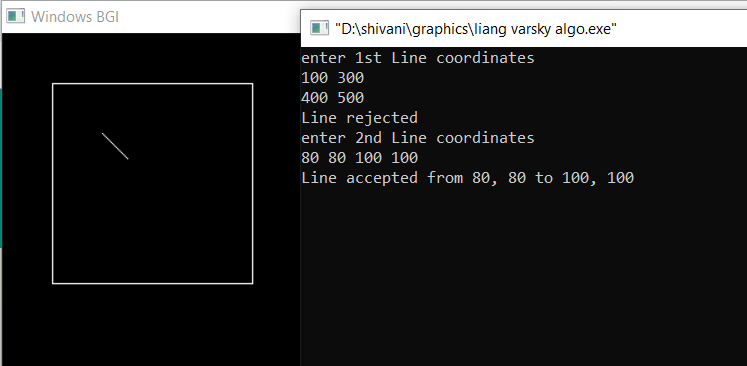
cohenSutherlandClip(x1,y1,x2,y2);

getch();

closegraph();

return 0;

}



**PROGRAM 12**

**LIANG BARSKY LINE CLIPPING ALFORITHM**

#include<iostream>

#include<conio.h>

#include<graphics.h>

using namespace std;

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

main()

{

int i,gd=DETECT,gm;

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

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

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

cout<<"enter the line coordinates:"<<endl;

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

cout<<"enter window coordinates:"<<endl;

cin>>xmin>>ymin>>xmax>>ymax;

rectangle(xmin,ymin,xmax,ymax);

dx=x2-x1;

dy=y2-y1;

p[0]=-dx;

p[1]=dx;

p[2]=-dy;

p[3]=dy;

q[0]=x1-xmin;

q[1]=xmax-x1;

q[2]=y1-ymin;

q[3]=ymax-y1;

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

{

if(p[i]==0)

{

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];

dda(xx1,yy1,xx2,yy2);

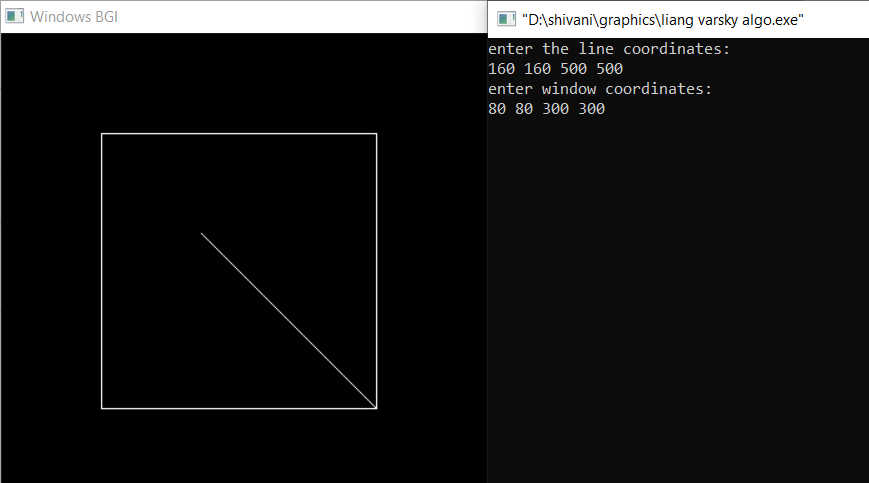
}

getch();

//delay(5000);

closegraph();

}



**PROGRAM 13**

**DRAW A HUT WITH 2 WINDOWS**

#include<graphics.h>

#include<conio.h>

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) steps;

    float X = X0;

    float Y = Y0;

    for (int i = 0; i <= steps; i++)

    {

        putpixel (X,Y,WHITE);

        X += Xinc;

        Y += Yinc;

    }

}

void bresenham\_circle(int xc,int yc,int r)

{

    int x=0;

    int y=r;

    int d=3-2\*r;

    while(y>=x)

    {

        x++;

            if(d<0)

            {

            d=d+4\*x+6;

            }

            else if(d>=0)

            {

            d=d+4\*x-4\*y+10;

            y=y-1;

            }

    putpixel(xc+x,yc+y,WHITE);

    putpixel(xc-x,yc+y,WHITE);

    putpixel(xc+x,yc-y,WHITE);

    putpixel(xc-x,yc-y,WHITE);

    putpixel(xc+y,yc+x,WHITE);

    putpixel(xc-y,yc+x,WHITE);

    putpixel(xc+y,yc-x,WHITE);

    putpixel(xc-y,yc-x,WHITE);

    }

}

void rectangle1(int x1,int y1,int x2,int y2)

{

    DDA(x1,y1,x2,y1);

    DDA(x1,y2,x2,y2);

    DDA(x2,y1,x2,y2);

    DDA(x1,y1,x1,y2);

}

int main(){

 int gd = DETECT,gm;

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

    setcolor(WHITE);

    rectangle1(150,180,250,300);

    rectangle1(250,180,420,300);

    rectangle1(180,250,220,300);

    rectangle1(300,200,350,250);

    dda(200,100,150,180);

    dda(200,100,250,180);

    dda(200,100,370,100);

    dda(370,100,420,180);

    bresenham\_circle(200,160,10);

 getch();

    closegraph();

    return 0;

}



**PROGRAM 14**

**DRAW A LAPTOP WITH SOME OBJECTS IN IT**

#include<iostream>

#include<graphics.h>

#include<math.h>

#include<conio.h>

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

void rectangle1(int x1,int y1,int x2,int y2)

{

dda(x1,y1,x2,y1);

dda(x1,y2,x2,y2);

dda(x2,y1,x2,y2);

dda(x1,y1,x1,y2);

}

int main(){

int gd = DETECT,gm;

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

rectanglel(100,100,320,260);

rectanglel(110,110,310,250);

dda(90,360,330,360);

dda(90,365,330,365);

dda(90,360,110,260);

dda(330,360,310,260);

dda(120,270,300,270);

dda(110,315,310,315);

dda(120,270,110,315);

dda(300,270,310,315);

dda(185,325,225,325);

dda(180,345,230,345);

dda(185,325,180,345);

dda(225,325,230,345);

setfillstyle(SOLID\_FILL, 7);

floodfill(111, 111, WHITE);

setfillstyle(HATCH\_FILL, GREEN);

floodfill(121, 271, WHITE);

setfillstyle(SOLID\_FILL, 15);

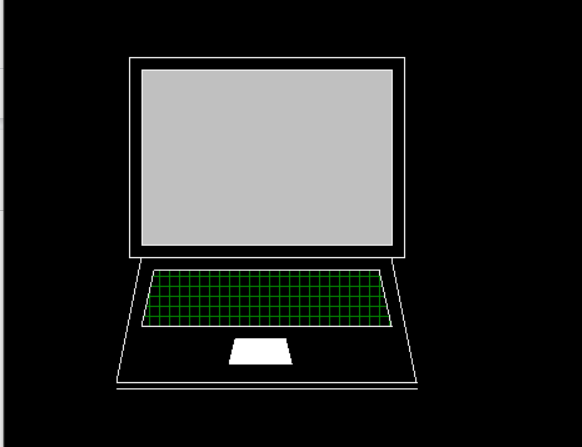
floodfill(186, 326, WHITE);

getch();

closegraph();

return 0;

}



**PROGRAM 15**

**DRAW A CEILING FAN**

#include<graphics.h>

#include<conio.h>

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

void midpoint\_circle(int r,int xc,int yc)

{

int x=0,y=r;

int p=1-r;

putpixel(x,y,WHITE);

while(x<y)

{

if(p<0)

{

p=p+2\*x+1;

x+=1;

}

else

{

p=p+2\*x-2\*y+1;

x+=1;

y--;

}

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc-y,WHITE);

putpixel(xc+y,yc+x,WHITE);

putpixel(xc-y,yc+x,WHITE);

putpixel(xc+y,yc-x,WHITE);

putpixel(xc-y,yc-x,WHITE);

}

}

main()

{

int gd = DETECT, gm;

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

midpoint\_circle(300,250,45);

dda(280,210,280,110);

dda(320,210,320,110);

dda(280,110,320,110);

dda(340,230,440,230);

dda(340,270,440,270);

dda(440,230,440,270);

dda(280,290,280,390);

dda(320,290,320,390);

dda(280,390,320,390);

dda(260,230,160,230);

dda(260,270,160,270);

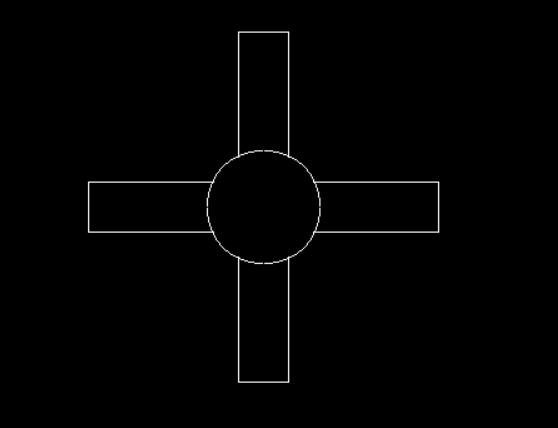
dda(160,230,160,270);

getch();

closegraph();

return 0;

}



**PROGRAM 16**

**DRAW A FULL MOON AND THREE STARS**

#include <graphics.h>

#include<conio.h>

void midpoint\_circle(int r,int xc,int yc)

{

int x=0,y=r;

int p=1-r;

putpixel(x,y,WHITE);

while(x<y)

{

if(p<0)

{

p=p+2\*x+1;

x+=1;

}

else

{

p=p+2\*x-2\*y+1;

x+=1;

y--;

}

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc-y,WHITE);

putpixel(xc+y,yc+x,WHITE);

putpixel(xc-y,yc+x,WHITE);

putpixel(xc+y,yc-x,WHITE);

putpixel(xc-y,yc-x,WHITE);

}

}

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

int main()

{

int gd = DETECT, gm;

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

int border\_color = WHITE;

setfillstyle(SOLID\_FILL,WHITE);

midpoint\_circle(50,70,70);

//floodfill(100,100,border\_color);

dda(300,300,400,300);

dda(300,300,350,213.4);

dda(400,300,350,213.4);

dda(290,257.73,410,257.73);

dda(290,257.73,350,335);

dda(410,257.73,350,335);

dda(500,300,600,300);

dda(500,300,550,213.4);

dda(600,300,550,213.4);

dda(490,257.73,610,257.73);

dda(490,257.73,550,335);

dda(610,257.73,550,335);

dda(300,100,400,100);

dda(300,100,350,13.4);

dda(400,100,350,13.4);

dda(290,57.73,410,57.73);

dda(290,57.73,350,135);

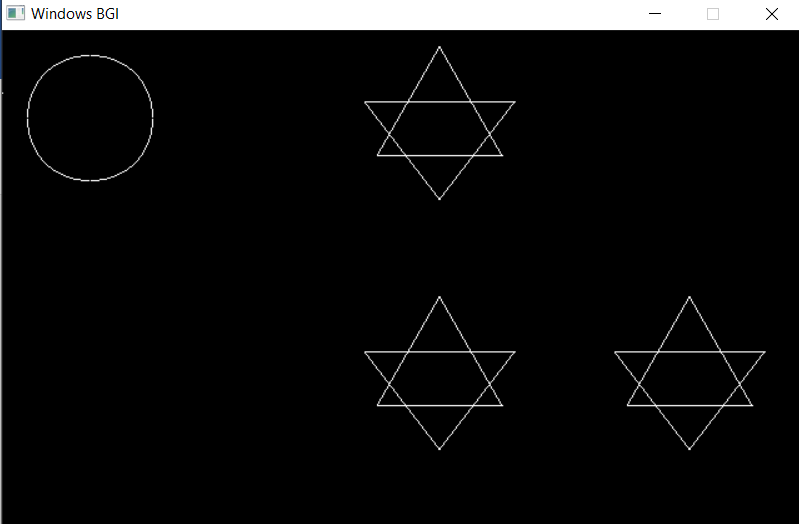
dda(410,57.73,350,135);

getch();

closegraph();

return 0;

}



**PROGRAM 17**

**DRAW A SCREEN AND CPU**

#include<graphics.h>

#include<conio.h>

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

void rect(int x1,int y1,int x2,int y2)

{

dda(x1,y1,x2,y1);

dda(x2,y1,x2,y2);

dda(x2,y2,x1,y2);

dda(x1,y1,x1,y2);

}

void midpoint\_circle(int r,int xc,int yc)

{

int x=0,y=r;

int p=1-r;

putpixel(x,y,WHITE);

while(x<y)

{

if(p<0)

{

p=p+2\*x+1;

x+=1;

}

else

{

p=p+2\*x-2\*y+1;

x+=1;

y--;

}

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc-y,WHITE);

putpixel(xc+y,yc+x,WHITE);

putpixel(xc-y,yc+x,WHITE);

putpixel(xc+y,yc-x,WHITE);

putpixel(xc-y,yc-x,WHITE);

}

}

void ellipse1(int rx,int ry,int xc,int yc)

{

float dx,dy,d1,d2,x,y;

x=0;

y=ry;

d1=(ry\*ry)-(rx\*rx\*ry)+(0.25\*rx\*rx); // Initial decision parameter of region 1

dx=2\*ry\*ry\*x;

dy=2\*rx\*rx\*y;

while(dx<dy) // For region 1

{

putpixel(x+xc,y+yc,WHITE);

putpixel(-x+xc,y+yc,WHITE);

putpixel(x+xc,-y+yc,WHITE);

putpixel(-x+xc,-y+yc,WHITE);

if(d1<0)

{

x++;

dx=dx+(2\*ry\*ry);

d1=d1+dx+(ry\*ry);

}

else

{

x++;

y--;

dx=dx+(2\*ry\*ry);

dy=dy-(2\*rx\*rx);

d1=d1+dx-dy+(ry\*ry);

}

}

d2=((ry\*ry)\*((x+0.5)\*(x+0.5)))+((rx\*rx)\*((y-1)\*(y-1)))-(rx\*rx\*ry\*ry);

while (y >= 0)

{

putpixel(x+xc,y+yc,WHITE);

putpixel(-x+xc,y+yc,WHITE);

putpixel(x+xc,-y+yc,WHITE);

putpixel(-x+xc,-y+yc,WHITE);

if(d2>0)

{

y--;

dy=dy-(2\*rx\*rx);

d2=d2+(rx\*rx)-dy;

}

else

{

y--;

x++;

dx=dx+(2\*ry\*ry);

dy=dy-(2\*rx\*rx);

d2=d2+dx-dy+(rx\*rx);

}

}

}

int main(){

int gd = DETECT,gm;

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

rect(100,50,320,200);

rect(110,60,310,190);

ellipse1(55,15,210,225);

rect(360,70,450,300);

rect(370,100,440,120);

midpoint\_circle(10,420,140);

putpixel(420,140,GREEN);

dda(200,200,200,220);

dda(220,200,220,220);

dda(110,260,310,260);

//line(90,360,330,360);

dda(90,320,110,260);

dda(330,320,310,260);

dda(330,320,90,320);

dda(120,270,300,270);

dda(110,315,310,315);

dda(120,270,110,315);

dda(300,270,310,315);

setfillstyle(SOLID\_FILL, BLUE);

floodfill(111, 111, WHITE);

floodfill(371, 101, WHITE);

setfillstyle(HATCH\_FILL, CYAN);

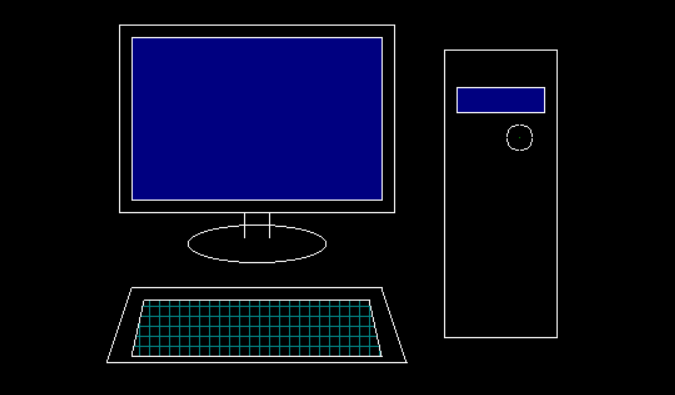
floodfill(121, 271, WHITE);

getch();

closegraph();

return 0;

}



**PROGRAM 18**

**DRAW A MOBILE PHONE**

#include<graphics.h>

#include<conio.h>

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

void rect(int x1,int y1,int x2,int y2)

{

dda(x1,y1,x2,y1);

dda(x2,y1,x2,y2);

dda(x2,y2,x1,y2);

dda(x1,y1,x1,y2);

}

void midpoint\_circle(int r,int xc,int yc)

{

int x=0,y=r;

int p=1-r;

putpixel(x,y,WHITE);

while(x<y)

{

if(p<0)

{

p=p+2\*x+1;

x+=1;

}

else

{

p=p+2\*x-2\*y+1;

x+=1;

y--;

}

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc-y,WHITE);

putpixel(xc+y,yc+x,WHITE);

putpixel(xc-y,yc+x,WHITE);

putpixel(xc+y,yc-x,WHITE);

putpixel(xc-y,yc-x,WHITE);

}

}

int main()

{

int gd = DETECT,gm;

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

rect(100,80,280,380);

setfillstyle(SOLID\_FILL, BLUE);

floodfill(111, 111, WHITE);

midpoint\_circle(15,190,400);

rect(185,395,195,405);

dda(90,50,90,400);

dda(290,50,290,400);

dda(110,30,270,30);

dda(110,420,270,420);

rect(150,65,230,68);

midpoint\_circle(5,190,57);

double x[4]={90,90,90,110},y[4]={50,30,30,30},px,py,i;

double t;

for(t=0.0;t<=1.0;t+=0.001){

px=(1-t)\*(1-t)\*(1-t)\*x[0]+3\*t\*(1-t)\*(1-t)\*x[1]+3\*t\*t\*(1-t)\*x[2]+t\*t\*t\*x[3];

py=(1-t)\*(1-t)\*(1-t)\*y[0]+3\*t\*(1-t)\*(1-t)\*y[1]+3\*t\*t\*(1-t)\*y[2]+t\*t\*t\*y[3];

putpixel(px,py,WHITE);

}

x[0]=270;

x[1]=291;

x[2]=291;

x[3]=291;

y[0]=30;

y[1]=30;

y[2]=30;

y[3]=50;

for(t=0.0;t<=1.0;t+=0.001){

px=(1-t)\*(1-t)\*(1-t)\*x[0]+3\*t\*(1-t)\*(1-t)\*x[1]+3\*t\*t\*(1-t)\*x[2]+t\*t\*t\*x[3];

py=(1-t)\*(1-t)\*(1-t)\*y[0]+3\*t\*(1-t)\*(1-t)\*y[1]+3\*t\*t\*(1-t)\*y[2]+t\*t\*t\*y[3];

putpixel(px,py,WHITE);

}

x[0]=90;

x[1]=90;

x[2]=90;

x[3]=110;

y[0]=400;

y[1]=421;

y[2]=421;

y[3]=421;

for(t=0.0;t<=1.0;t+=0.001){

px=(1-t)\*(1-t)\*(1-t)\*x[0]+3\*t\*(1-t)\*(1-t)\*x[1]+3\*t\*t\*(1-t)\*x[2]+t\*t\*t\*x[3];

py=(1-t)\*(1-t)\*(1-t)\*y[0]+3\*t\*(1-t)\*(1-t)\*y[1]+3\*t\*t\*(1-t)\*y[2]+t\*t\*t\*y[3];

putpixel(px,py,WHITE);

}

x[0]=291;

x[1]=291;

x[2]=291;

x[3]=270;

y[0]=400;

y[1]=421;

y[2]=421;

y[3]=421;

for(t=0.0;t<=1.0;t+=0.001){

px=(1-t)\*(1-t)\*(1-t)\*x[0]+3\*t\*(1-t)\*(1-t)\*x[1]+3\*t\*t\*(1-t)\*x[2]+t\*t\*t\*x[3];

py=(1-t)\*(1-t)\*(1-t)\*y[0]+3\*t\*(1-t)\*(1-t)\*y[1]+3\*t\*t\*(1-t)\*y[2]+t\*t\*t\*y[3];

putpixel(px,py,WHITE);

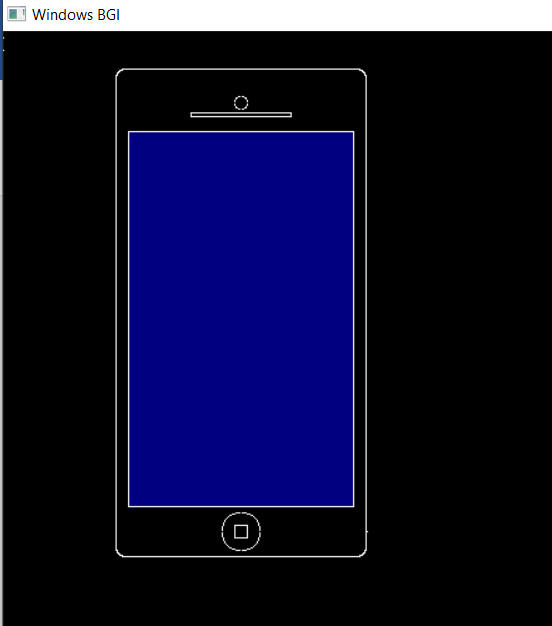
}

getch();

closegraph();

return 0;

}



**PROGRAM 19**

**DRAW A BAR GRAPH HAVING ATLEAST THREE BARS**

#include<iostream>

#include<graphics.h>

using namespace std;

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

int main()

{

int gd = DETECT,gm;

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

DDA(100,300,100,100);

DDA(100,300,300,300);

DDA(125,300,125,200);

DDA(135,300,135,200);

DDA(125,200,135,200);

DDA(150,300,150,250);

DDA(160,300,160,250);

DDA(150,250,160,250);

DDA(175,300,175,150);

DDA(185,300,185,150);

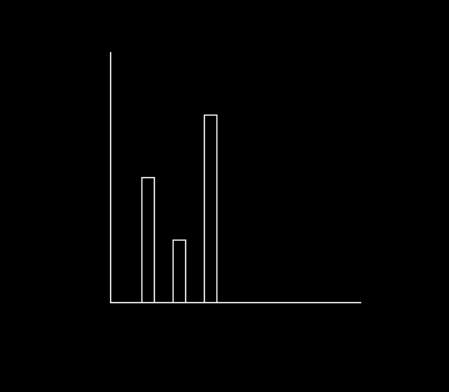
DDA(175,150,185,150);

getch();

closegraph();

return 0;

}



**PROGRAM 20**

**DRAW A MOVING CAR**

#include <graphics.h>

#include <iostream>

#include<math.h>

using namespace std;

void circle1(int r,int xc,int yc)

{

int x=0,y=r;

int p=1-r;

putpixel(x,y,WHITE);

while(x<y)

{

if(p<0)

{

p=p+2\*x+1;

x+=1;

}

else

{

p=p+2\*x-2\*y+1;

x+=1;

y--;

}

putpixel(xc+x,yc+y,WHITE);

putpixel(xc-x,yc+y,WHITE);

putpixel(xc+x,yc-y,WHITE);

putpixel(xc-x,yc-y,WHITE);

putpixel(xc+y,yc+x,WHITE);

putpixel(xc-y,yc+x,WHITE);

putpixel(xc+y,yc-x,WHITE);

putpixel(xc-y,yc-x,WHITE);

}

}

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) steps;

float X = X0;

float Y = Y0;

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

{

putpixel (X,Y,WHITE);

X += Xinc;

Y += Yinc;

}

}

void draw\_moving\_car(void) {

int i, j = 0, gd = DETECT, gm;

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

while(1){

for (i = 10; i <= 400; i = i + 10) {

setcolor(WHITE);

if(210+i>600){

i-=10;

break;

}

rectangle(10,400,600,50);

dda(0 + i, 300, 210 + i, 300);

dda(50 + i, 300, 75 + i, 270);

dda(75 + i, 270, 150 + i, 270);

dda(150 + i, 270, 165 + i, 300);

dda(0 + i, 300, 0 + i, 330);

dda(210 + i, 300, 210 + i, 330);

circle1(15,65 + i, 330);

circle1(2,65 + i, 330);

circle1(15,145 + i, 330);

circle1(2,145 + i, 330);

dda(0 + i, 330, 50 + i, 330);

dda(80 + i, 330, 130 + i, 330);

dda(210 + i, 330, 160 + i, 330);

delay(50);

cleardevice();

}

for(;i>=10;i-=10){

rectangle(10,400,600,50);

dda(0 + i, 300, 210 + i, 300);

dda(50 + i, 300, 75 + i, 270);

dda(75 + i, 270, 150 + i, 270);

dda(150 + i, 270, 165 + i, 300);

dda(0 + i, 300, 0 + i, 330);

dda(210 + i, 300, 210 + i, 330);

circle1(15,65 + i, 330);

circle1(2,65 + i, 330);

circle1(15,145 + i, 330);

circle1(2,145 + i, 330);

dda(0 + i, 330, 50 + i, 330);

dda(80 + i, 330, 130 + i, 330);

dda(210 + i, 330, 160 + i, 330);

delay(50);

cleardevice();

}

}

getch();

closegraph();

}

int main()

{

draw\_moving\_car();

return 0;

}

**PROGRAM 21**

**A BOUNCING BALL**

#include <stdio.h>

#include <conio.h>

#include <graphics.h>

#include <dos.h>

int main() {

int gd = DETECT, gm;

int i, x, y, flag=0;

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

x = getmaxx()/2;

y = 30;

while (!kbhit()) {

if(y >= getmaxy()-30 || y <= 30)

flag = !flag;

setcolor(RED);

setfillstyle(SOLID\_FILL, RED);

circle(x, y, 30);

floodfill(x, y, RED);

delay(50);

cleardevice();

if(flag){

y = y + 5;

} else {

y = y - 5;

}

}

getch();

closegraph();

return 0;

}