**DR B.R. AMBEDKAR NATIONAL INSTITUTE OF**

**TECHNOLOGY JALANDHAR,**

**PUNJAB, INDIA**



**Computer Graphics**

**And**

**Animation Laboratory**

**CSX-328**

**Session: Jan-May 2020**

**SUBMITTED BY-**

Name – Ankit Goyal

Roll no - 17103011

Group - G-1

Branch – CSE

**SUBMITTED TO-**

Mr. Rahul Aggarwal

Assistant Professor

CSE Department

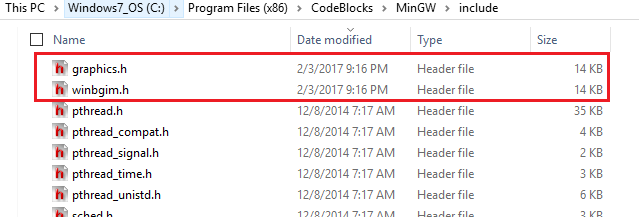
**INDEX**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Practical** | **Page** | **Remarks** |
|  | To setup graphics.h library with Code::Blocks. | 3-4 |  |
|  | DDA Line Algorithm | 5-6 |  |
|  | Bresenham Algorithm | 7-10 |  |
|  | Polynomial Circle | 11-12 |  |
|  | Trignometric Circle | 13-14 |  |
|  | Mid-point Circle | 15-16 |  |
|  | Bresenham Circle | 17-18 |  |
|  | Polynomial Ellipse | 19 |  |
|  | Mid-point Ellipse | 20 |  |
|  | Trignometric Ellipse | 21-22 |  |
|  | Translation of a triangle | 23 |  |
|  | Scaling of a triangle | 24-25 |  |
|  | Rotation of an Ellipse with angle thete | 26-28 |  |
|  | Translation, rotation and scaling of a rectangle | 29-32 |  |
|  | Implement Bezier Curve | 33-34 |  |
|  | Implementation of​ ​ Liang Basky Line Clipping Algorithm. | 35-37 |  |

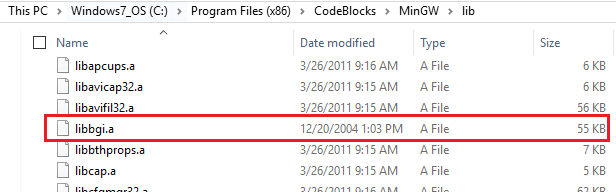
**Practical No. 1**

**Aim: To setup graphics.h library with Code::blocks..**

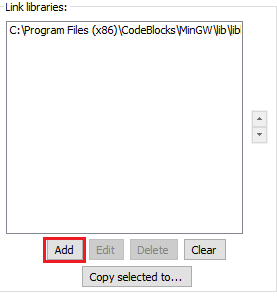
1. Firstly download the graphics.h and winbgim.h library from <http://winbgim.codecutter.org/> and copy them in the folder C:\Program Files (x86)\CodeBlocks\MinGW\include .



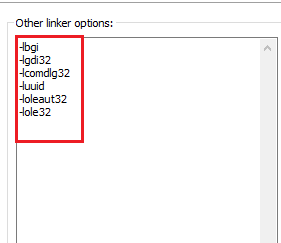
1. Now download the libbgi.a file from <http://winbgim.codecutter.org/> and copy them in the folder C:\Program Files (x86)\CodeBlocks\MinGW\lib.



1. In Code::Blocks open Settings >> Compiler >>Linker settings click Add button in Link Libraries part and browse and select libbgi.a file from the path C:\Program Files (x86)\CodeBlocks\MinGW\lib.



1. In Code::Blocks open Settings >> Compiler >>Linker settings in right part (i.e. other linker options) paste commands -lbgi -lgdi32 -lcomdlg32 -luuid -loleaut32 -lole32



**Practical No. 2**

**Aim: Write a program for DDA Line Drawing Algorithm.**

**Program:**

#include<graphics.h>

#include<bits/stdc++.h>

using namespace std;

int main()

{

int gd = DETECT ,gm, i=1;

float x, y,dx,dy,steps;

int x0, x1, y0, y1;

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

cout<<"enter two points:(x1,y1,x2,y2) ";

cin>>x0>>y0>>x1>>y1;

dx = (float)(x1 - x0);

dy = (float)(y1 - y0);

if(dx>=dy)

steps = dx;

else

steps = dy;

dx = dx/steps;

dy = dy/steps;

x = x0;

y = y0;

while(i<= steps)

{

putpixel(x, y, 15);

x += dx;

y += dy;

i=i+1;

}

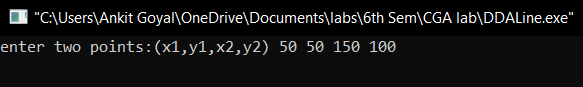
getch();

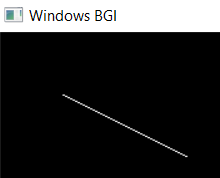
closegraph();

return 0;

}

**Output:**

****

****

**Practical No. 3**

**Aim: Write a program for Bresenham Line Drawing Algorithm.**

**Program:**

#include<bits/stdc++.h>

#include<graphics.h>

using namespace std;

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

{

int dx,dy;

int x,y,x\_end,y\_end;

dx = (x2-x1);

dy = (y2-y1);

float m = dy/dx;

dx = abs(dx);

dy = abs(dy);

if(abs(m)<=1 && m>0)

{

cout<<1;

int p = 2\*dy-dx;

if(x1>x2)

{

x = x2;

x\_end = x1;

y = y2;

}

else

{

x = x1;

x\_end = x2;

y = y1;

}

while(x<=x\_end)

{

putpixel(x,y,WHITE);

x = x+1;

if(p<0)

p = p + 2\*dy;

else

{

y = y+1;

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

}

}

}

else if(abs(m)<=1 && m<0)

{

cout<<2;

int p = 2\*dy-dx;

if(x1>x2)

{

x = x2;

x\_end = x1;

y = y2;

}

else

{

x = x1;

x\_end = x2;

y = y1;

}

while(x<=x\_end)

{

putpixel(x,y,WHITE);

x = x+1;

if(p<0)

p = p + 2\*dy;

else

{

y = y-1;

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

}

}

}

else if(abs(m)>=1 && m>0)

{

cout<<3;

int p = 2\*dx-dy;

if(y1>y2)

{

y = y2;

y\_end = y1;

x= x2;

}

else

{

x = x1;

y\_end = y2;

y = y1;

}

while(y<=y\_end)

{

putpixel(x,y,WHITE);

y = y+1;

if(p<0)

p = p + 2\*dx;

else

{

x= x+1;

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

}

}

}

else

{

cout<<4;

int p = 2\*dx-dy;

if(y1>y2)

{

y = y2;

y\_end = y1;

x= x2;

}

else

{

x = x1;

y\_end = y2;

y = y1;

}

while(y<=y\_end)

{

putpixel(x,y,WHITE);

y = y+1;

if(p<0)

p = p + 2\*dx;

else

{

x= x-1;

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

}

}

}

}

int main()

{

int gdriver=DETECT, gmode, error, x0, y0, x1, y1;

cout<<"enter two points:(x1,y1,x2,y2) ";

cin>>x0>>y0;

cin>>x1>>y1;

initgraph(&gdriver, &gmode, "");

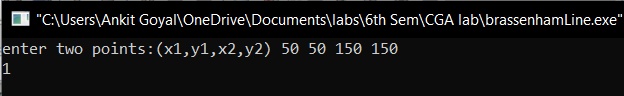
drawline(x0, y0, x1, y1);

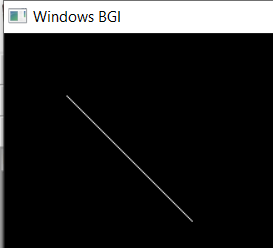
getch();

return 0;

}

**Output:**

****

****

**Practical No. 4**

**Aim: Write a program for Trigonometric Circle Drawing Algorithm.**

**Program:**

#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

#define color 15

using namespace std;

void eightSymmetricPointsPlot(int xc,int yc,int x,int y)

{

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

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

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

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

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

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

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

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

}

int main(){

float xc,yc,p,x1,y1,y,r,x;

float theta;

int gd=DETECT,gm;

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

cout<<"Enter the center of circle:";

cin>>xc>>yc;

cout<<"Enter radius of circle:";

cin>>r;

float i=0;

while(i<=45){

theta=(i\*3.14)/180;

x=(r\*cos(theta));

y=(r\*sin(theta));

eightSymmetricPointsPlot(xc,yc,x,y);

i=i+0.01;

}

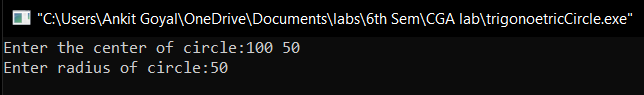
getch();

closegraph();

return 0;

}

**Output:**

****

****

**Practical No. 5**

**Aim: Write a program for Polynomial Circle Drawing Algorithm.**

**Program:**

#include<graphics.h>

#include<conio.h>

#include<math.h>

#include<bits/stdc++.h>

using namespace std;

void setPixel(int x, int y, int h, int k)

{

putpixel(x+h, y+k, 15);

putpixel(x+h, -y+k, 15);

putpixel(-x+h, -y+k, 15);

putpixel(-x+h, y+k, 15);

putpixel(y+h, x+k, 15);

putpixel(y+h, -x+k, 15);

putpixel(-y+h, -x+k, 15);

putpixel(-y+h, x+k, 15);

}

main()

{

int gd=0, gm,h,k,r;

double x,y,x2;

cout<<"enter center: ";

cin>>h>>k;

cout<<"enter radius: ";

cin>>r;

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

setbkcolor(WHITE);

x=0,y=r;

x2 = r/sqrt(2);

while(x<=x2)

{

y = sqrt(r\*r - x\*x);

setPixel(floor(x), floor(y), h,k);

x += 1;

}

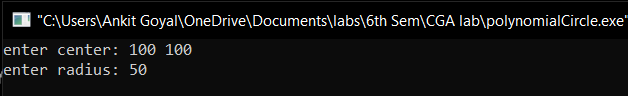
getch();

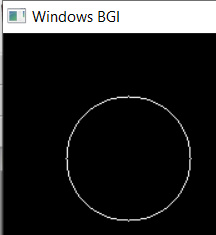
closegraph();

return 0;

}

**Output:**

****

****

**Practical No. 6**

**Aim: Write a program for Bresenham Circle Drawing Algorithm.**

**Program:**

#include <graphics.h>

#include <bits/stdc++.h>

#define color 15

using namespace std;

void eightWaySymmetricPlot(int xc,int yc,int x,int y)

{

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

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

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

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

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

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

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

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

}

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

{

int x,y,d;

x=0;

y=r;

d=3-2\*r;

eightWaySymmetricPlot(xc,yc,x,y);

while(x<=y)

{

if(d<=0)

d=d+4\*x+6;

else

{

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

y=y-1;

}

x=x+1;

eightWaySymmetricPlot(xc,yc,x,y);

}

}

int main()

{

int gdriver = DETECT, gmode;

int xc,yc,r;

initgraph(&gdriver, &gmode, "");

cout<<"Enter center point of circle : ";

cin>>xc>>yc;

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

cin>>r;

BressanhamCircle(xc,yc,r);

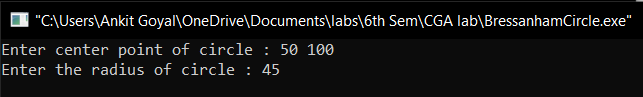
getch();

closegraph();

return 0;

}

**Output:**

****

****

**Practical No. 7**

**Aim: Write a program for Mid-Point Circle Drawing Algorithm.**

**Program:**

#include<iostream>

#include<bits/stdc++.h>

#include<graphics.h>

#define color 15

using namespace std;

void eightSymmetricPointsPlot(int xc,int yc,int x,int y)

{

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

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

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

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

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

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

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

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

}

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

{

int x,y,d;

x=0;

y=r;

d=1-r;

eightSymmetricPointsPlot(xc,yc,x,y);

while(x<=y)

{

if(d<=0)

{

d=d+2\*x+10;

}

else

{

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

y=y-1;

}

x=x+1;

eightSymmetricPointsPlot(xc,yc,x,y);

}

}

int main()

{

int gdriver = DETECT, gmode;

int xc,yc,r;

initgraph(&gdriver, &gmode, "");

cout<<"Enter center point of circle : ";

cin>>xc>>yc;

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

cin>>r;

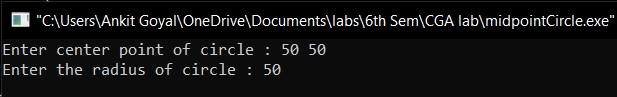
midPointCircle(xc, yc, r);

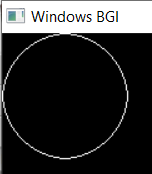
getch();

return 0;

}

**Output:**

****

****

**Practical No. 8**

**Aim: Write a program for Trigonometric Ellipse Drawing Algorithm.**

**Program:**

#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

using namespace std;

void plot4pixels(int x,int y,int h,int k)

{

putpixel(x+h,y+k,8);

putpixel(x+h,-y+k,8);

putpixel(-x+h,y+k,8);

putpixel(-x+h,-y+k,8);

}

int main(){

float xc,yc,y,a,b,x;

float theta;

int gd=DETECT,gm;

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

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

cin>>xc>>yc;

cout<<"Enter both axes of ellipse:";

cin>>a>>b;

float i=0;

while(i<=90){

theta=(i\*3.14)/180;

x=(a\*cos(theta));

y=(b\*sin(theta));

plot4pixels(x,y,xc,yc);

i=i+0.01;

}

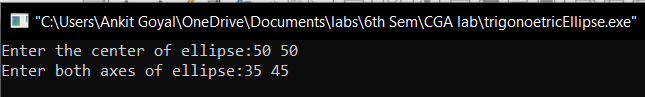
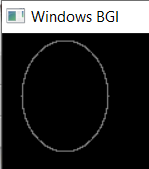
getch();

closegraph();

return 0;

}

**Output:**

** **

**Practical No. 9**

**Aim: Write a program for Polynomial Ellipse Drawing Algorithm.**

**Program:**

#include<iostream>

#include<graphics.h>

#include<conio.h>

#include<math.h>

using namespace std;

void plot4pixels(int x,int y,int h,int k)

{

putpixel(x+h,y+k,8);

putpixel(x+h,-y+k,8);

putpixel(-x+h,y+k,8);

putpixel(-x+h,-y+k,8);

}

int main()

{

int x,y,r,i,h,k,a,b;

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

cin>>h>>k;

cout<<"Enter both axes of ellipse:";

cin>>a>>b;

x=0;

y=b;

int gd=DETECT,gm;

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

setbkcolor(WHITE);

while(x<a)

{

plot4pixels(x,y,h,k);

x++;

y=b\*sqrt(((a\*a)-(x\*x\*1.0))/(a\*a));

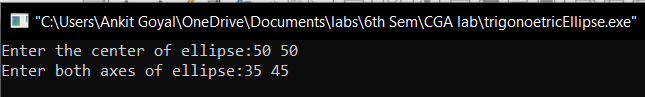
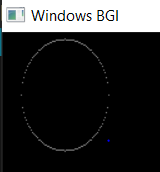
}

plot4pixels(x,y,h,k);

getch();

}

**Output:**

****

**Practical No. 10**

**Aim: Write a program for Mid-Point Ellipse Drawing Algorithm.**

**Program:**

#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

using namespace std;

void plot4pixels(int x,int y,int h,int k)

{

putpixel(x+h,y+k,8);

putpixel(x+h,-y+k,8);

putpixel(-x+h,y+k,8);

putpixel(-x+h,-y+k,8);

}

void midptellipse(int a, int b, int xc, int yc)

{

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

x = 0;

y = b;

d1 = (b \* b) - (a \* a \* b) + (0.25 \* a \* a);

dx = 2 \* b \* b \* x;

dy = 2 \* a \* a \* y;

while (dx < dy)

{

plot4pixels(x,y,xc,yc);

if (d1 < 0)

{

x++;

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

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

}

else

{

x++;

y--;

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

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

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

}

}

d2 = ((b \* b) \* ((x + 0.5) \* (x + 0.5))) +

((a \* a) \* ((y - 1) \* (y - 1))) -

(a \* a \* b \* b);

while (y >= 0)

{

plot4pixels(x,y,xc,yc);

if (d2 > 0)

{

y--;

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

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

}

else

{

y--;

x++;

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

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

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

}

}

}

int main(){

float xc,yc,y,a,b,x;

float theta;

int gd=DETECT,gm;

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

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

cin>>xc>>yc;

cout<<"Enter both axes of ellipse:";

cin>>a>>b;

midptellipse(a,b,xc,yc);

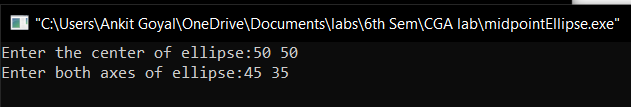
getch();

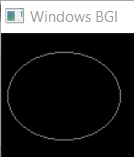
closegraph();

return 0;

}

**Output:**

****

****

**Practical No. 11**

**Aim: To translate an object with translation parameters in X and Y directions.**

**Program:**

#include<iostream>

#include<graphics.h>

using namespace std;

void scale(float x[], float y[], float sx, float sy)

{

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

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

{

x[i]+=sx;

y[i]+=sy;

}

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

}

int main()

{

int gd = DETECT, gm;

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

float x[3],y[3],sx,sy;

cout<<"enter the vertexes\n";

cin>>x[0]>>y[0]>>x[1]>>y[1]>>x[2]>>y[2];

cout<<"enter translation factors\n";

cin>>sx>>sy;

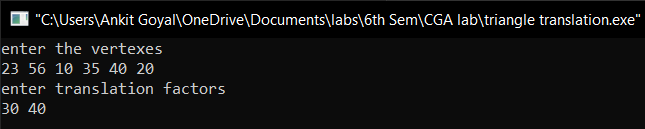
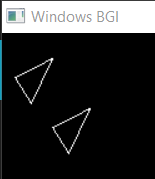
scale(x, y, sx,sy);

getch();

closegraph();

}

**Output:**

** **

**Practical No. 12**

**Aim: To translate an object with scaling factors along X and Y directions.**

**Program:**

#include<iostream>

#include<graphics.h>

using namespace std;

void findNewCoordinate(float s[][2], float p[][1])

{

float temp[2][1] = { 0 };

for (int i = 0; i < 2; i++)

for (int j = 0; j < 1; j++)

for (int k = 0; k < 2; k++)

temp[i][j] += (s[i][k] \* p[k][j]);

p[0][0] = temp[0][0];

p[1][0] = temp[1][0];

}

void scale(float x[], float y[], float sx, float sy)

{

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

float s[2][2] = { sx, 0, 0, sy };

float p[2][1];

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

{

p[0][0] = x[i];

p[1][0] = y[i];

findNewCoordinate(s, p);

x[i] = p[0][0];

y[i] = p[1][0];

}

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[0], y[0]);

}

int main()

{

int gd = DETECT, gm;

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

float x[3],y[3],sx,sy;

cout<<"enter the vertexes\n";

cin>>x[0]>>y[0]>>x[1]>>y[1]>>x[2]>>y[2];

cout<<"enter scaling factors\n";

cin>>sx>>sy;

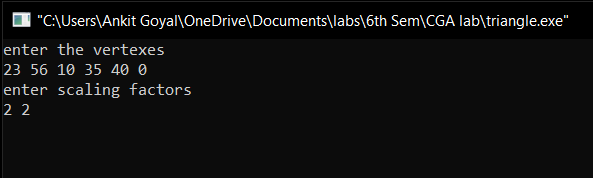
scale(x, y, sx,sy);

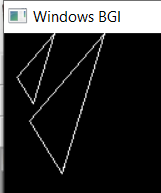
getch();

closegraph();

}

**Output:**

****

****

**Practical No. 13**

**Aim: To Draw an ellipse and rotate it through an angle theta.**

**Program:**

#include<iostream>

#include<conio.h>

#include<graphics.h>

#include<math.h>

#include<bits/stdc++.h>

using namespace std;

vector<double> v[3];

void plot4pixels(int x,int y,int h,int k)

{

putpixel(x+h,y+k,8);

v[0].push\_back(x+h);

v[1].push\_back(y+k);

v[2].push\_back(1);

putpixel(x+h,-y+k,8);

v[0].push\_back(x+h);

v[1].push\_back(-y+k);

v[2].push\_back(1);

putpixel(-x+h,y+k,8);

v[0].push\_back(-x+h);

v[1].push\_back(y+k);

v[2].push\_back(1);

putpixel(-x+h,-y+k,8);

v[0].push\_back(-x+h);

v[1].push\_back(-y+k);

v[2].push\_back(1);

}

void midptellipse(int a, int b, int xc, int yc)

{

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

x = 0;

y = b;

d1 = (b \* b) - (a \* a \* b) + (0.25 \* a \* a);

dx = 2 \* b \* b \* x;

dy = 2 \* a \* a \* y;

while (dx < dy)

{

plot4pixels(x,y,xc,yc);

if (d1 < 0)

{

x++;

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

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

}

else

{

x++;

y--;

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

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

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

}

}

d2 = ((b \* b) \* ((x + 0.5) \* (x + 0.5))) +

((a \* a) \* ((y - 1) \* (y - 1))) -

(a \* a \* b \* b);

while (y >= 0)

{

plot4pixels(x,y,xc,yc);

if (d2 > 0)

{

y--;

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

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

}

else

{

y--;

x++;

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

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

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

}

}

}

int main(){

float xc,yc,y,a,b,x,temp;

double theta;

int gd=DETECT,gm;

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

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

cin>>xc>>yc;

cout<<"Enter both axes of ellipse:";

cin>>a>>b;

cout<<"enter angle:";

cin>>theta;

double R\_mat[3][3]={{cos(theta),-sin(theta),0},{sin(theta),cos(theta),0},{0,0,1}};

midptellipse(a,b,xc,yc);

vector<double> res[3];

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

{

for(int j=0;j<v[0].size();j++)

{

temp=0;

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

temp+=R\_mat[i][k]\*v[k][j];

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

}

}

for(int i=0;i<res[0].size();i++)

putpixel(res[0][i],res[1][i],8);

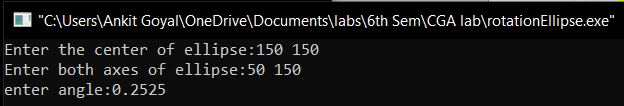
getch();

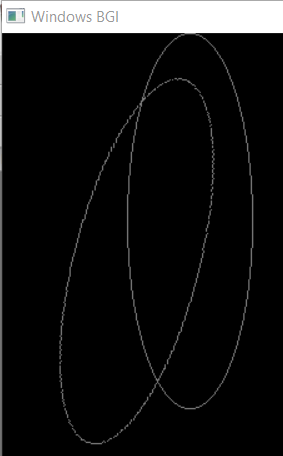
closegraph();

return 0;

}

**Output:**

****



**Practical No. 14**

**Aim: To Draw an Rectangle and translate, rotate and scale it.**

**Program:**

#include<bits/stdc++.h>

#include<graphics.h>

using namespace std;

void findNewCoordinate(int s[][2], int p[][1])

{

int temp[2][1] = { 0 };

for (int i = 0; i < 2; i++)

for (int j = 0; j < 1; j++)

for (int k = 0; k < 2; k++)

temp[i][j] += (s[i][k] \* p[k][j]);

p[0][0] = temp[0][0];

p[1][0] = temp[1][0];

}

void findNewCoordinate(float s[][2], int p[][1])

{

float temp[2][1] = { 0 };

for (int i = 0; i < 2; i++)

for (int j = 0; j < 1; j++)

for (int k = 0; k < 2; k++)

temp[i][j] += (s[i][k] \* p[k][j]);

p[0][0] = temp[0][0];

p[1][0] = temp[1][0];

}

void findNewCoordinates(int s[][3], int p[][1])

{

int temp[3][1] = { 0 };

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

for (int j = 0; j < 1; j++)

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

temp[i][j] += (s[i][k] \* p[k][j]);

p[0][0] = temp[0][0];

p[1][0] = temp[1][0];

}

void scale(int x[], int y[], int sx, int sy)

{

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[3], y[3]);

line(x[3] , y[3] ,x[0] , y[0] );

int s[2][2] = { sx, 0, 0, sy };

int p[2][1];

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

{

p[0][0] = x[i];

p[1][0] = y[i];

findNewCoordinate(s, p);

x[i] = p[0][0];

y[i] = p[1][0];

}

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[3], y[3]);

line(x[3] , y[3] , x[0] , y[0] );

}

void translate(int x[], int y[], int tx, int ty)

{

int s[3][3] = { 1, 0, tx, 0 , 1 , ty , 0 , 0 , 1 };

int p[3][1];

int n=3;

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

{

p[0][0] = x[i];

p[1][0] = y[i];

p[2][0] = 1;

findNewCoordinates(s, p);

x[i] = p[0][0];

y[i] = p[1][0];

}

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[3], y[3]);

line(x[3] , y[3] , x[0] , y[0] );

}

void rotation(int x[] , int y[] , float theta)

{

float c = cos(theta);

float se = sin(theta);

float s[2][2] = { c, -se, se, c };

int p[2][1];

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

{

p[0][0] = x[i];

p[1][0] = y[i];

findNewCoordinate(s, p);

x[i] = p[0][0];

y[i] = p[1][0];

}

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[3], y[3]);

line(x[3] , y[3] , x[0] , y[0] );

}

int main()

{

int y[4] ,x[4] ;

cout<<"Enter the coordinates of four points\n";

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

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

int gd, gm ,sx , sy;

detectgraph(&gd, &gm);

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

line(x[0], y[0], x[1], y[1]);

line(x[1], y[1], x[2], y[2]);

line(x[2], y[2], x[3], y[3]);

line(x[3] , y[3] ,x[0] , y[0] );

float theta;

int tx , ty;

cout<<"1. For translation\n2. For Scaling\n3. For rotation\n enter your choice\n ";

int ch;

cin>>ch;

switch(ch)

{

case 1:

{

cout<<"Enter the x and y distance for translation\n";

cin>>tx>>ty;

translate(x , y , tx, ty);

break;

}

case 2:

{

cout<<"Enter the x and y scaling factor\n";

cin>>sx>>sy;

scale(x , y, sx , sy);

break;

}

case 3:

{

cout<<"Enter the angle for rotation in radian\n";

cin>>theta;

rotation(x , y , theta);

break;

}

default :

cout<<"enter a valid choice\n";

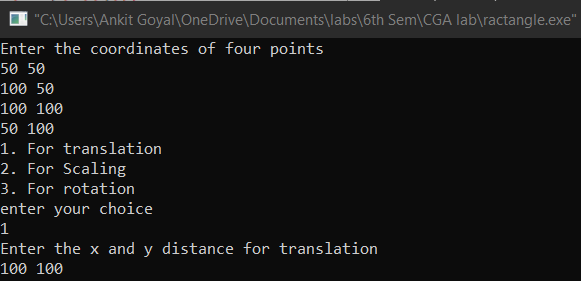
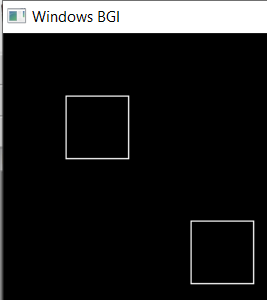
}

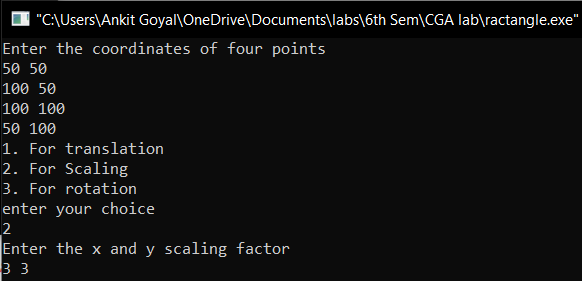
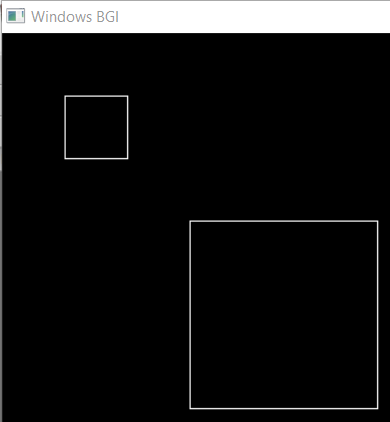
getch();

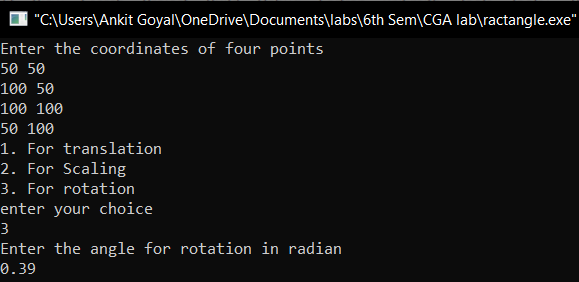
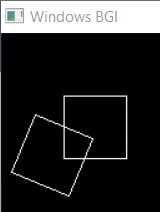
return 0;

}

**Output:**

**** 

** **

**** 

**Practical No. 15**

**Aim: Write a program to implement Bezier Curve.**

**Program:**

include<iostream.h>

#include<conio.h>

#include<math.h>

#include<graphics.h>

void main()

{

int gd=DETECT,gm;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

int x[4],y[4],i;

double put\_x, put\_y, t;

cout<<"\n\*\*\*\*\*\* Bezier Curver \*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Enter four control points of bezier curve: "<<endl;

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

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

putpixel(x[i], y[i],3);

}

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

put\_x=pow(1-t, 3)\*x[0]+3\*t\*pow(1-t,2)\*x[1]+3\*t\*t\*(1-t)\*x[2]+pow(t,3)\*x[3];

put\_y=pow(1-t, 3)\*y[0]+3\*t\*pow(1-t,2)\*y[1]+3\*t\*t\*(1-t)\*y[2]+pow(t,3)\*y[3];

putpixel(put\_x,put\_y,WHITE);

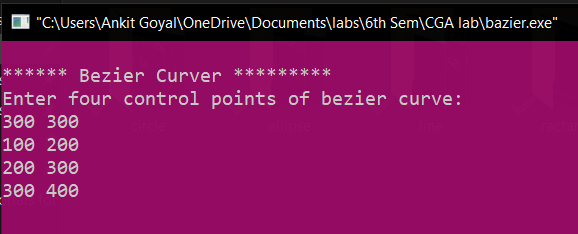
}

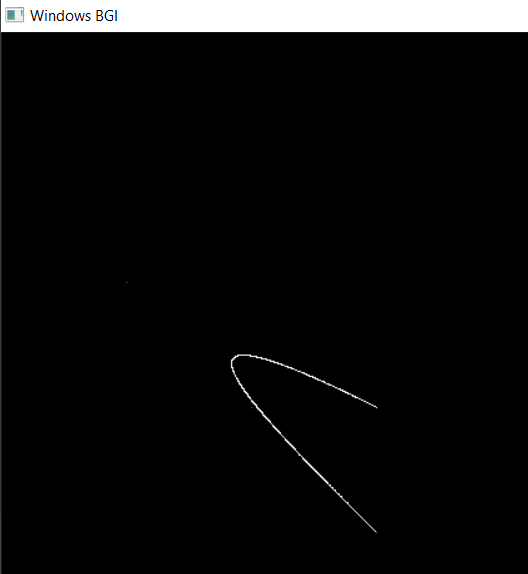
getch();

closegraph();

}

**Output:**

****

****

**Practical No. 16**

**Aim: Write a program to implement Liang Basky Line Clipping Algorithm.**

**Program:**

#include<iostream.h>

#include<graphics.h>

#include<math.h>

#include<dos.h>

void main()

{

int i,gd=DETECT,gm;

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

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

x1=120;

y1=120;

x2=300;

y2=300;

xmin=100;

ymin=100;

xmax=250;

ymax=250;

initgraph(&gd, &gm, "C:\\TURBOC3\\BGI");

rectangle(xmin,ymin,xmax,ymax);

dx=x2-x1;

dy=y2-y1;

p[0]=-dx;

p[1]=dx;

p[2]=-dy;

p[3]=dy;

q[0]=x1-xmin;

q[1]=xmax-x1;

q[2]=y1-ymin;

q[3]=ymax-y1;

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

{

if(p[i]==0)

{

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();

}

**Output:**

