Computer Graphics

Practical File

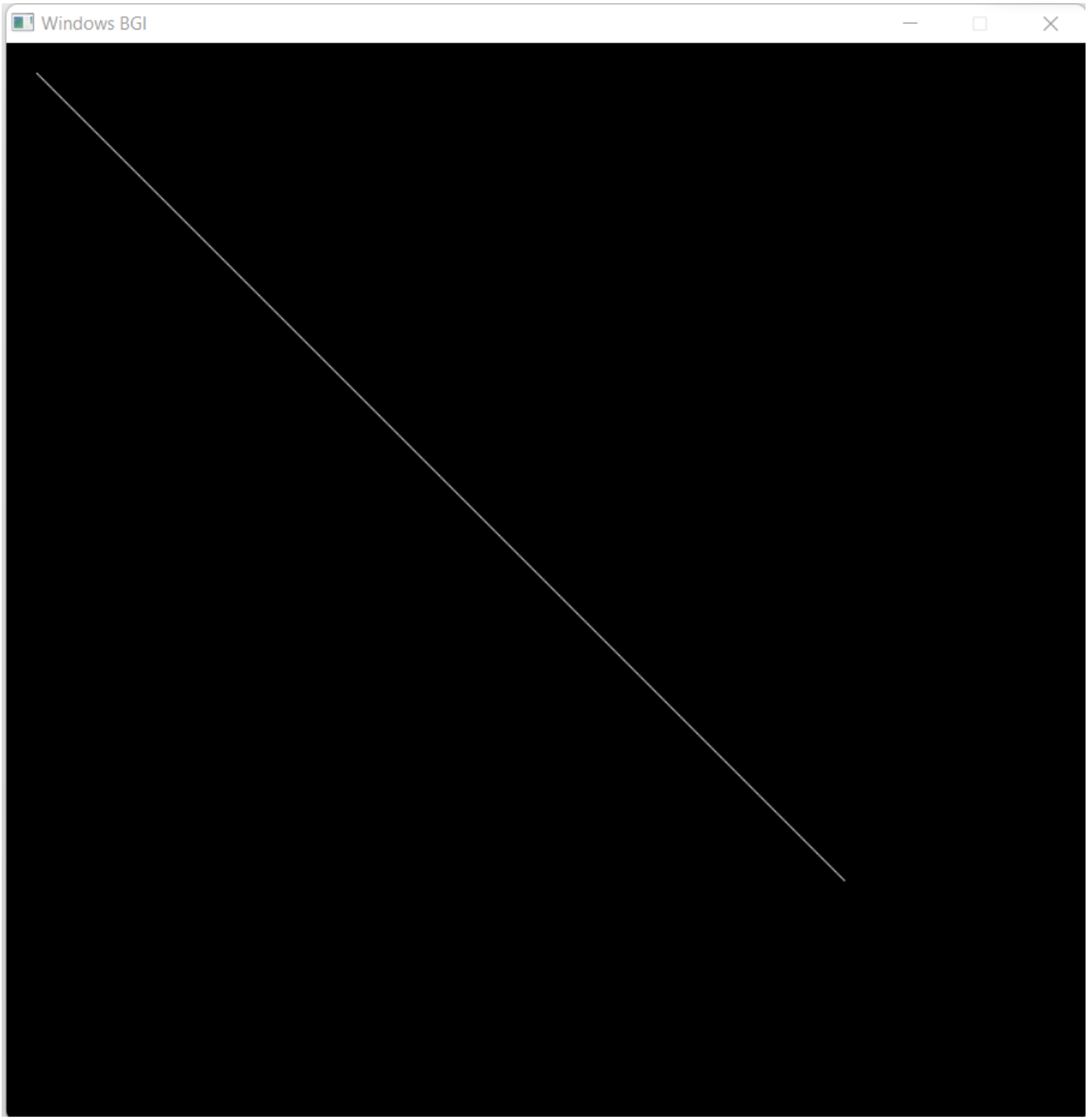
Submitted by:
Nishant Goel 2019UCO1529

Table of Content

1 .Generating line primitives using DDA

```
#// C program for DDA line generation

#include <graphics.h>
#include <math.h>
#include <stdio.h>
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)steps;
    float X = X0;
    float Y = Y0;
    for (int i = 0; i <= steps; i++) {
        putpixel(round(X), round(Y),
                WHITE);
        X  +=  Xinc;
        Y  +=  Yinc;
        delay(100);
    }
}

// Driver program
int main()
{
    int gd = DETECT, gm;
    initwindow(720, 720);
    DDA(20, 20, 700, 700);
    return 0;
}
```

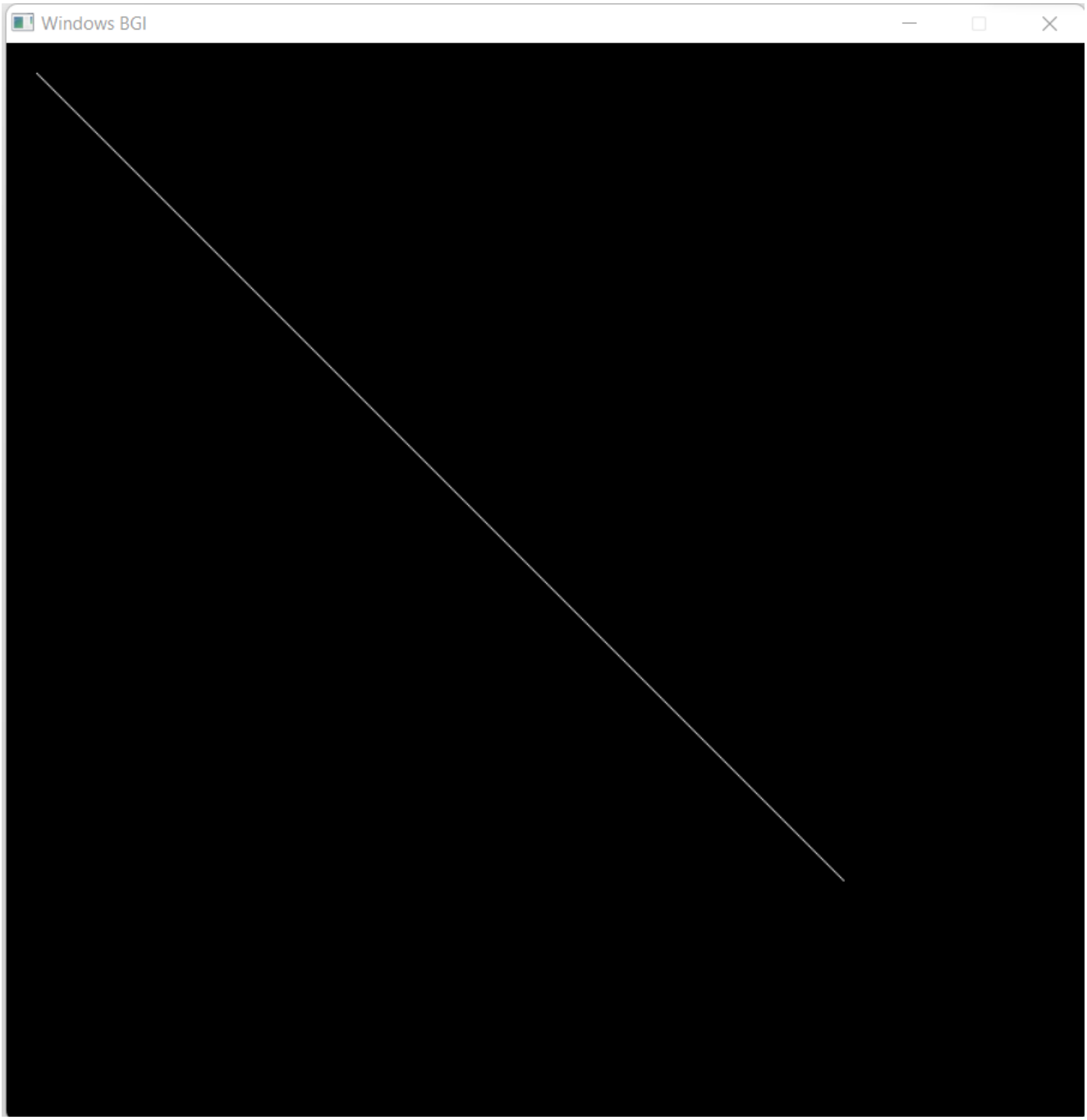
Output :



2.) Generating line using bresenham algorithm

```
// C++ program for Mid-point line generation
#include <bits/stdc++.h>
using namespace std;
#include <graphics.h>
// // Driver program
void lm(int x1, int y1, int x2, int y2){
    float m = float (y2-y1)/(x2-x1);
    float p = 1/2 - m;
    int x = x1, y = y1;
    putpixel(x, y, WHITE);
    while(x!=x2){
        if(p<0){
            p = p - m +1;
            y = y+1;
            x = x+1;
        }else{
            p = p - m;
            x = x+1;
        }
        putpixel(x, y, WHITE);
        delay(10);
    }

}

int main()
{
    initwindow(720, 720);
    int x1=100, y1=200, x2= 2000, y2=4000;
    lm(x1, y1, x2, y2);
    return 0;
}
```

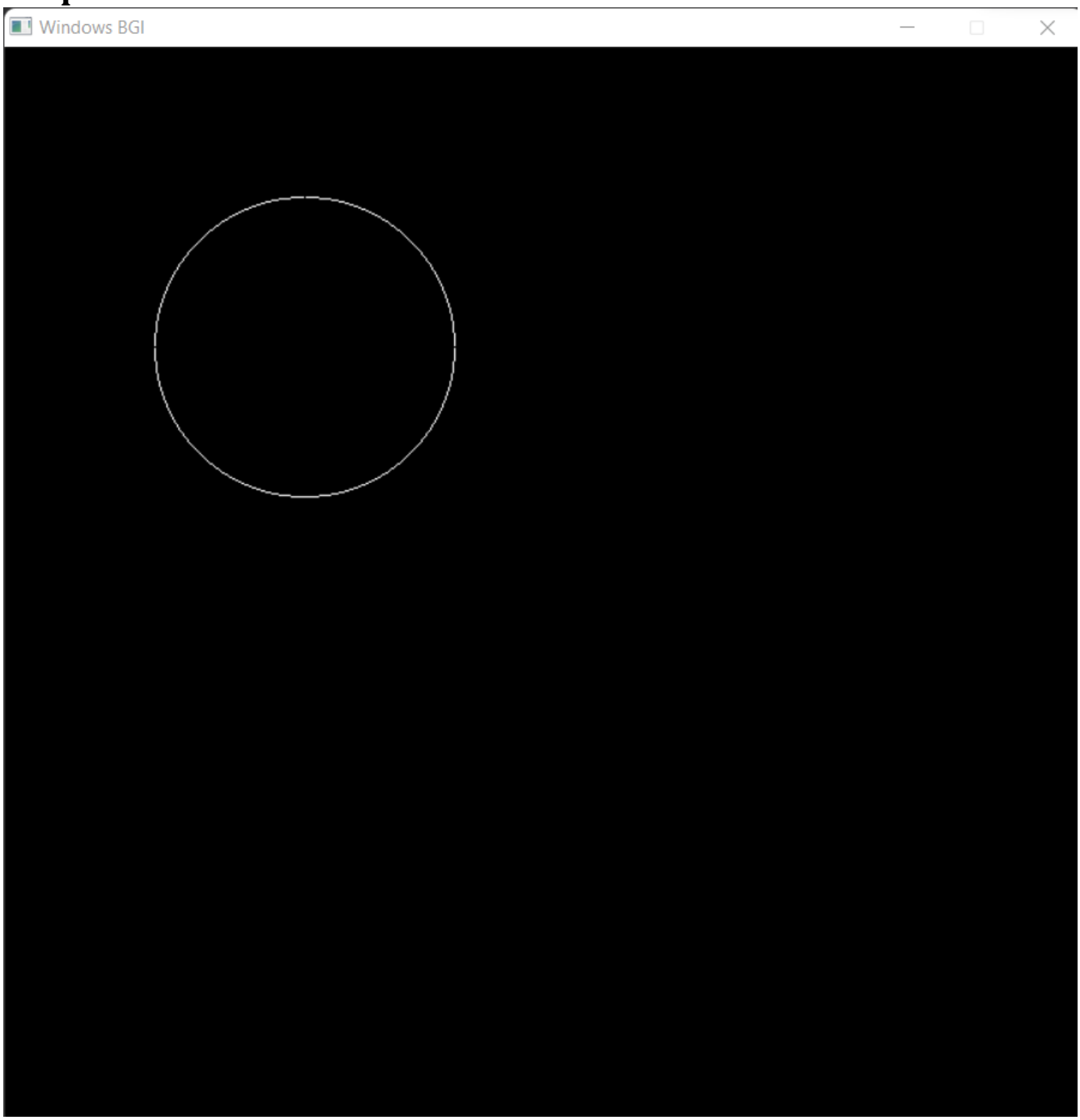
Output:


Windows BGI

3.) Generating circle using bresenham ’ s approach

```
#include<bits/stdc++.h>
using namespace std;
#include<graphics.h>
void draw(int x, int y){
    putpixel(x+200, y+200, WHITE);
    delay(1);
    putpixel(x+200, -y+200, WHITE);
    delay(1);
    putpixel(-x+200, -y+200, WHITE);
    delay(1);
    putpixel(-x+200, y+200, WHITE);
    delay(1);
    putpixel(y+200, x+200, WHITE);
    delay(1);
    putpixel(y+200, -x+200, WHITE);
    delay(1);
    putpixel(-y+200, x+200, WHITE);
    delay(1);
    putpixel(-y+200, -x+200, WHITE);

}
void circle1(int a, int b, int c){
    float p =3 - 2*c;
    int x=0, y=c;
    cout<<x<<", "<<y<<"\n";
    putpixel(x+a, y+b, WHITE);
    while(x<=y){
        if(p<0){
            p = p + 4*x +6;
            x = x+1;
            draw(x, y);
        }else{
            p = p + 4*(x-y) + 10;
            x = x+1;
            y = y-1;
            draw(x, y);
        }
    }
}
int main()
{
    initwindow(720,720);
    int a=200, b=200, c= 100;
    circle1(a, b, c);
    return 0;
    closegraph();
}
```

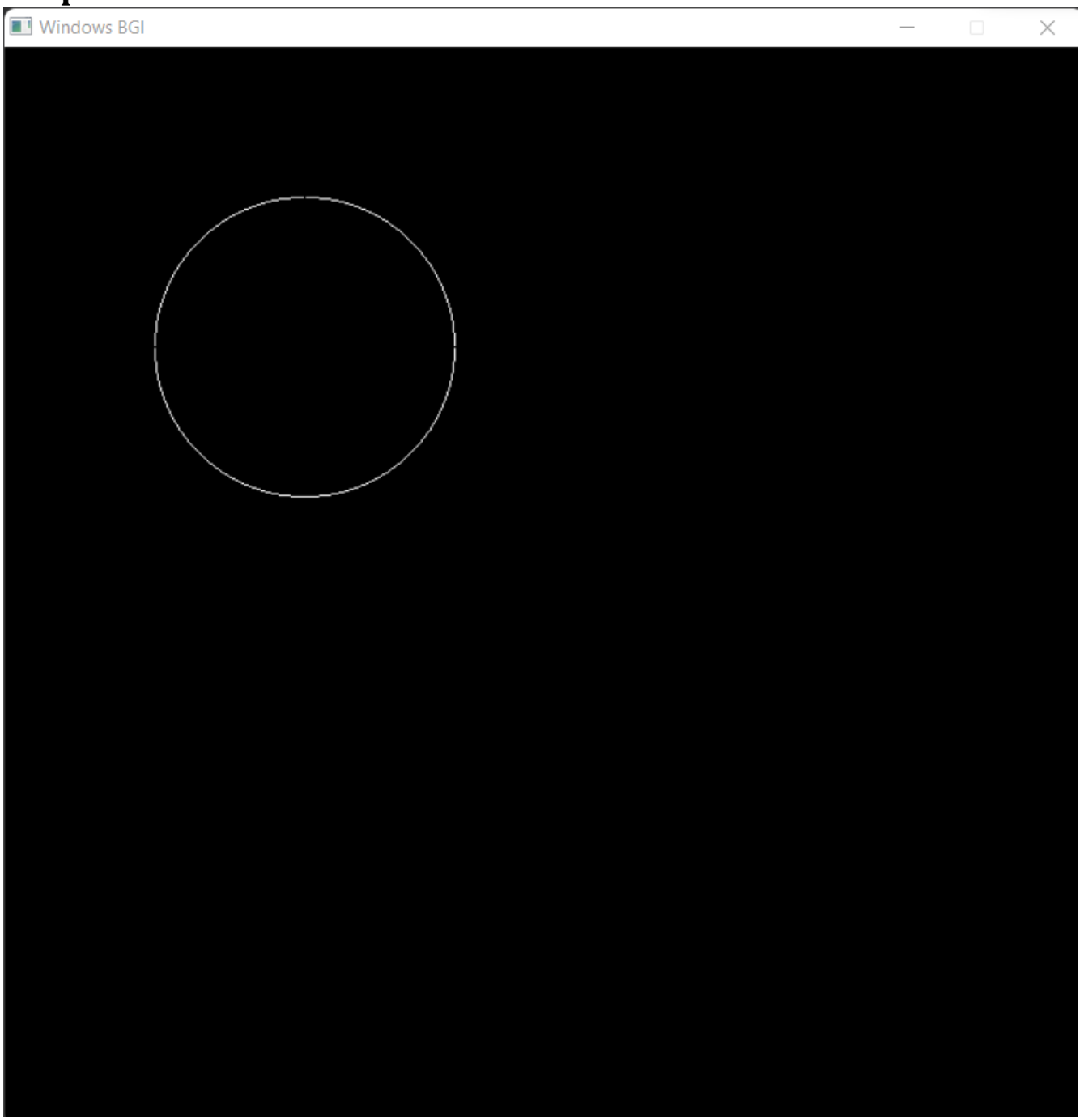
Output:



4.) Generating circle using Mid point approach

```
#include<bits/stdc++.h>
using namespace std;
#include<graphics.h>
void draw(int x, int y){
    putpixel(x+200, y+200, WHITE);
    delay(1);
    putpixel(x+200, -y+200, WHITE);
    delay(1);
    putpixel(-x+200, -y+200, WHITE);
    delay(1);
    putpixel(-x+200, y+200, WHITE);
    delay(1);
    putpixel(y+200, x+200, WHITE);
    delay(1);
    putpixel(y+200, -x+200, WHITE);
    delay(1);
    putpixel(-y+200, x+200, WHITE);
    delay(1);
    putpixel(-y+200, -x+200, WHITE);

}
void circle1(int a, int b, int c){
    float p =3 - 2*c;
    int x=0, y=c;
    cout<<x<<", "<<y<<"\n";
    putpixel(x+a, y+b, WHITE);
    while(x<=y){
        if(p<0){
            p = p + 4*x +6;
            x = x+1;
            draw(x, y);
        }else{
            p = p + 4*(x-y) + 10;
            x = x+1;
            y = y-1;
            draw(x, y);
        }
    }
}
int main()
{
    initwindow(720,720);
    int a=200, b=200, c= 100;
    circle1(a, b, c);
    return 0;
    closegraph();
}
```

Output:

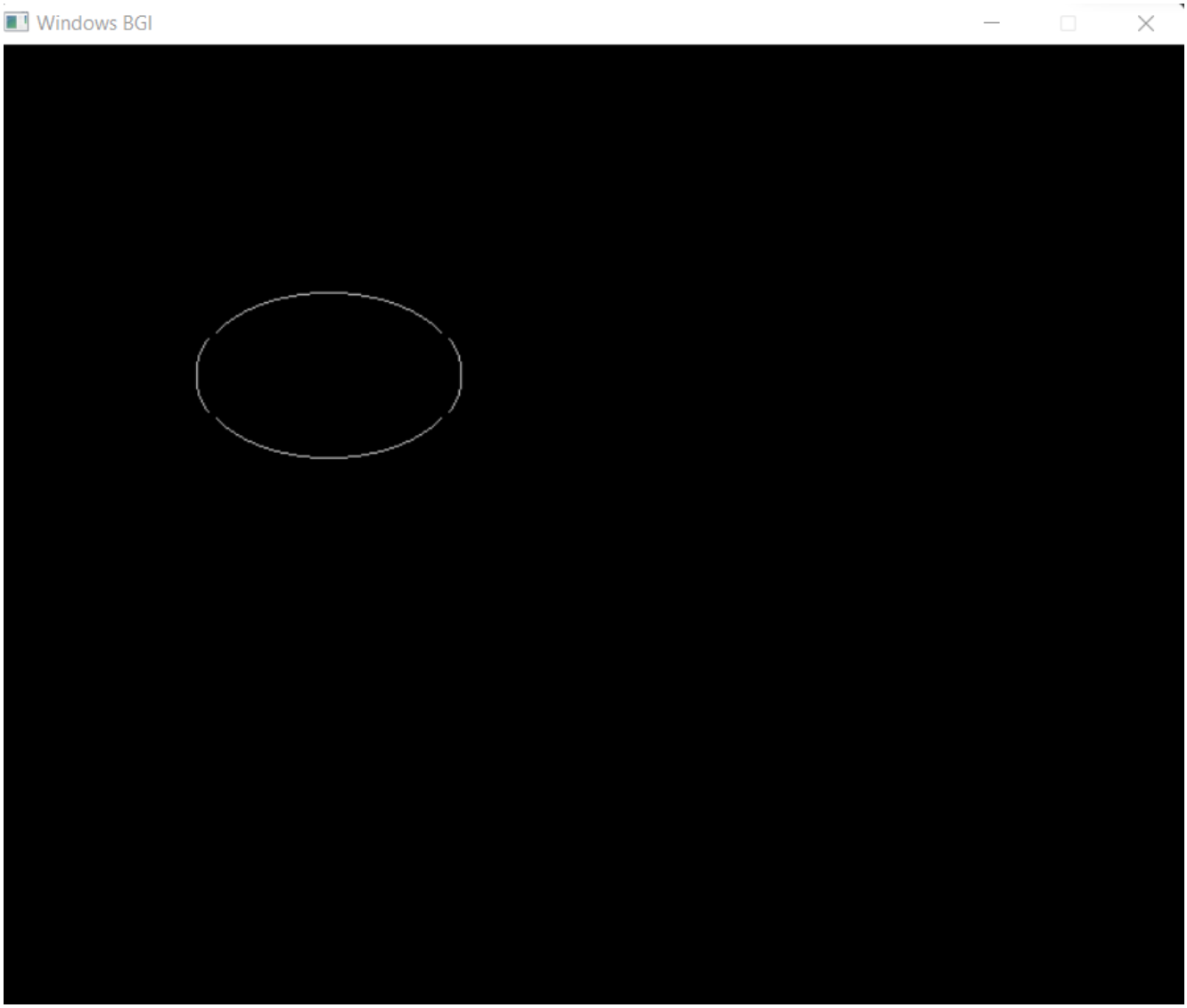


5.) Generating ellipse using mid point algorithm

```
#include <stdio.h>
#include <conio.h>
#include <graphics.h>
#include <math.h>
void disp();
float  x,  y;
int  xc,  yc;
int main(){

   float p1, p2;

   initwindow(720,720);
   int a, b;
   printf("*** Ellipse Generating Algorithm ***\n");
   printf("Enter the value of Xc\t");
   scanf("%d", &xc);
   printf("Enter the value of yc\t");
   scanf("%d", &yc);
   printf("Enter X axis length\t");
   scanf("%d",  &a);
   printf("Enter Y axis length\t");
   scanf("%d", &b);
   x = 0;
   y = b;
   disp();
   p1 = (b * b) - (a * a * b) + (a * a) / 4;
   while ((2.0 * b * b * x) <= (2.0 * a * a * y))
   {
      x++;
      if (p1 <= 0)
         p1 = p1 + (2.0 * b * b * x) + (b * b);
      else
      {
         y--;
         p1 = p1 + (2.0 * b * b * x) + (b * b) - (2.0 * a * a * y);
      }
      disp();
      x = -x;
      disp();
      x = -x;
      delay(50);
   }
   x = a;
   y = 0;
   disp();
   p2 = (a * a) + 2.0 * (b * b * a) + (b * b) / 4;
   while ((2.0 * b * b * x) > (2.0 * a * a * y))
   {
      y++;
```

```
        if (p2 > 0)
            p2 = p2 + (a * a) - (2.0 * a * a * y);
        else
        {
            x--;
            p2 = p2 + (2.0 * b * b * x) - (2.0 * a * a * y) + (a * a);
        }
        disp();
        y = -y;
        disp();
        y = -y;
        delay(50);
    }
    closegraph();
    return 0;
}
void disp()
{
    putpixel(xc + x, yc + y, 7);
    putpixel(xc - x, yc + y, 7);
    putpixel(xc + x, yc - y, 7);
    putpixel(xc + x, yc - y, 7);
}
```

Output:



6.) Generating hyperbola using Mid point algorithm

```
// C++ program for Mid-point line generation
#include <bits/stdc++.h>
using namespace std;
#include <graphics.h>
// // Driver program
void draw(int x, int y)
{
    putpixel(x + 200, y + 200, GREEN);
    delay(1);
    putpixel(x + 200, -y + 200, GREEN);
    delay(1);
     putpixel(-x + 200, -y + 200, GREEN);
    delay(1);
     putpixel(-x + 200, y + 200, GREEN);
```

```
    delay(1);
}
void hyp(int a, int b)
{
     double p = (float)(1/4 + a)*(b*b) - (float)1*(a*a);
     cout<<p<<"\n";
     int x= a, y=0;
     putpixel(x+200, y+200, GREEN);
     while(y< (b*b)/(sqrt(a*a - b*b))){
        if(p>0){
            p = p - (2*y+3)*(a*a);
            y = y+1;
            draw(x, y);
        }else{
            p = p + (2*(x+1)*(b*b)) - (2*y+3)*(a*a);
            y = y+1;
            x = x+1;
            draw(x, y);
        }
         cout<<p<<"\n";
     }
     p = (x + 1)*(x+1)*b*b - (y+1/2)*(y+1/2)*a*a - a*a*b*b;
     while(y<300){
        cout<<"aa";
        if(p>0){
            p = p + (2*x+3)*b*b - a*a*(2*(y+1));
            y=y+1;
            x = x+1;
            draw(x, y);
        }else{
            p = p + (2*x+3)*b*b;
            x = x+1;
            draw(x, y);
        }
     }
}

int main()
{
    initwindow(720, 720);
    int a = 40, b=20;
    hyp(a, b);
    return 0;
}
```

Output:

7. Implement Line Clipping approach using Cohen Sutherland

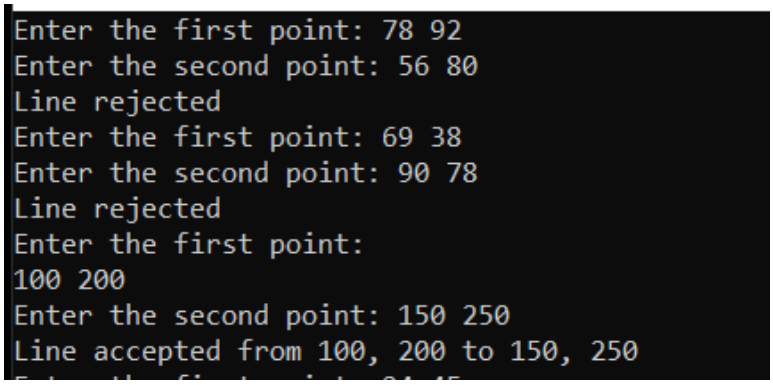
```
#include  <iostream>
#include <GL/glut.h>
using namespace std;

// Defining region codes
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 = 700;
const int y_max = 500;
const int x_min = 100;
const int y_min = 100;

void drawLine(int x1, int y1, int x2, int y2) {
   glBegin(GL_LINES);
   glVertex2i(x1, y1);
   glVertex2i(x2, y2);
   glEnd();
   glFlush();
}

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;
```

```
}

void lc_cs(){
   int x1, y1, x2, y2;
   cout << "Enter the first point: ";
   cin >> x1 >> y1;
   cout << "Enter the second point: ";
   cin >> x2 >> 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)) {
         accept = true;
         break;
      }
      else if (code1 & code2) {
         break;
      }
      else {
         int code_out;
         double x, y;
         if (code1 != 0)
            code_out = code1;
         else
            code_out = code2;
         if (code_out & TOP) {
            x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
            y = y_max;
         }
         else if (code_out & BOTTOM) {
            x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
            y = y_min;
```

```
            }
            else if (code_out & RIGHT) {
               y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
               x = x_max;
            }
            else if (code_out & LEFT) {
               y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
               x = x_min;
            }
            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;
         drawLine(x1, y1, x2, y2);
         // Here the user can add code to display the rectangle
         // along with the accepted (portion of) lines
      }
      else
         cout << "Line rejected" << endl;
}

int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition(100, 200);
   glutInitWindowSize(800, 600);
   glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");
   glClearColor(1, 1, 1, 1);
```

```
    glColor3f(0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    gluOrtho2D(0, 800, 0, 600);
    glutDisplayFunc(lc_cs);
    glutMainLoop();
}
```

Output :



8. Implement Line Clipping approach using Liang Barsky / Cyrus Beck

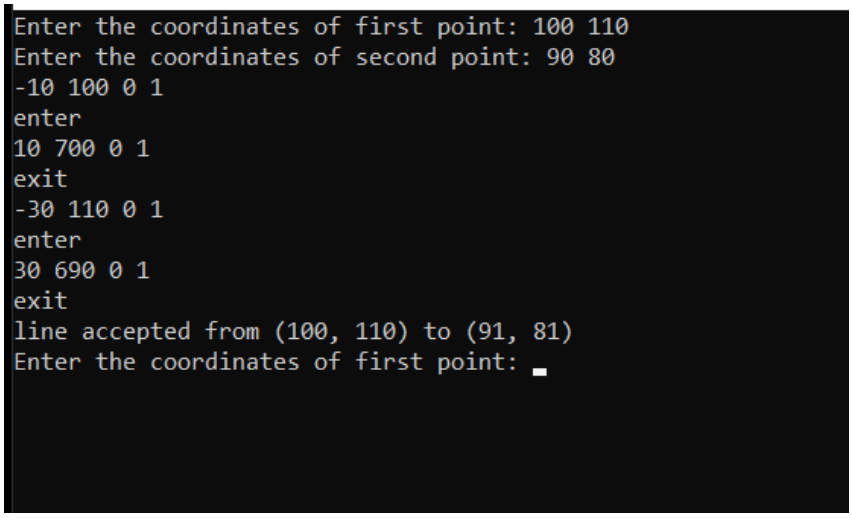
```
#include  <iostream>
#include <GL/glut.h>
using namespace std;

// Defining region codes
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 = 700;
const int y_max = 500;
const int x_min = 100;
const int y_min = 100;

void drawLine(int x1, int y1, int x2, int y2) {
   glBegin(GL_LINES);
   glVertex2i(x1, y1);
   glVertex2i(x2, y2);
   glEnd();
   glFlush();
}

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;
}

void lc_cs(){
   int x1, y1, x2, y2;
   cout << "Enter the first point: ";
   cin >> x1 >> y1;
   cout << "Enter the second point: ";
   cin >> x2 >> 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)) {
         accept = true;
         break;
      }
      else if (code1 & code2) {
         break;
      }  else {
         int code_out;
         double x, y;
         if (code1 != 0)
            code_out = code1;
         else
            code_out = code2;
         if (code_out & TOP) {
            x = x1 + (x2 - x1) * (y_max - y1) / (y2 - y1);
            y = y_max;
         } else if (code_out & BOTTOM) {
            x = x1 + (x2 - x1) * (y_min - y1) / (y2 - y1);
            y = y_min;
         } else if (code_out & RIGHT) {
```

```
            y = y1 + (y2 - y1) * (x_max - x1) / (x2 - x1);
            x = x_max;
         }
         else if (code_out & LEFT) {
            y = y1 + (y2 - y1) * (x_min - x1) / (x2 - x1);
            x = x_min;
         }
         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;
      drawLine(x1, y1, x2, y2);
      // Here the user can add code to display the rectangle
      // along with the accepted (portion of) lines
   } else
      cout << "Line rejected" << endl;
}
int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowPosition(100, 200);
   glutInitWindowSize(800, 600);
   glutCreateWindow("Cohen Sutherland Line Clipping Algorithm");
   glClearColor(1, 1, 1, 1);
   glColor3f(0, 0, 0);
   glClear(GL_COLOR_BUFFER_BIT);
   gluOrtho2D(0, 800, 0, 600);
   glutDisplayFunc(lc_cs);
```

```
    glutMainLoop();
}
```

Output :



9. Implement Line Clipping approach using Mid-Point Subdivision

```
#include  <iostream>
#include <GL/glut.h>
using namespace std;

#define XWMIN 100
#define XWMAX 400
#define YWMIN 100
#define YWMAX 400

void drawLine(int x1, int y1, int x2, int y2) {
   glBegin(GL_LINES);
   glVertex2i(x1, y1);
   glVertex2i(x2, y2);
   glEnd();
   glFlush();
}

void drawWindow(int xmin, int ymin, int xmax, int ymax) {
   glBegin(GL_LINE_LOOP);
   glVertex2i(xmin,  ymin);
   glVertex2i(xmin,  ymax);
   glVertex2i(xmax, ymax);
   glVertex2i(xmax,  ymin);
   glEnd();
   glFlush();
}

int calcCode(int x, int y){
   int code = 0;
   if(x < XWMIN) {
      //left
      code |= 1;
   }
 else if(x > XWMAX) { //right
      code |= 2;
   }
   if(y < YWMIN) {  //top
```

```
      code |= 4;
   }
   else if(y > YWMAX) { //bottom
      code |= 8;
   }
   return code;
}

void clipLine(int& xc1, int& yc1, int &xc2, int &yc2, int x1, int y1, int x2, int y2) {
   int xc11, yc11, xc12, yc12, xc21, yc21, xc22, yc22;
   int code1 = calcCode(x1, y1), code2 = calcCode(x2,y2);
   if(x1 == (x1 + x2)/2 && y1 == (y1 + y2)/2) {
      xc1 = x1;
      xc2 = x2;
      yc1 = y1;
      yc2 = y2;
      return;
   }
   if((code1 | code2) == 0) {
      //completely inside
      xc1 = x1; yc1 = y1; xc2 = x2; yc2 = y2;
      return;
   }
   else if((code1 & code2) != 0) {
      //completely outside
      xc1 = -1;
      yc1 = -1;
      xc2 = -1;
      yc2 = -1;
      return;
   }
   //clipping candidate
   clipLine(xc11, yc11, xc21, yc21, x1, y1, (x1 + x2)/2, (y1 + y2)/2);
   clipLine(xc12, yc12, xc22, yc22, (x1 + x2)/2, (y1 + y2)/2, x2, y2);
   if(xc21 == xc12 && yc21 == yc12) {
      xc1 = xc11;
      yc1 = yc11;
      xc2 = xc22;
```

```
      yc2 = yc22;
   }
   else if(xc11 == -1 && xc21 == -1 && yc11 == -1 && yc21 == -1) { //first point
invalid
      xc1 = xc12;
      xc2 = xc22;
      yc1 = yc12;
      yc2 = yc22;
   }
   else {
      //second point invalid
      xc1 = xc11;
      xc2 = xc21;
      yc1 = yc11;
      yc2 = yc21;
   }
}
void mpsd() {
   int x0, y0, x1, y1;
   cout << "Enter first point: ";
   cin >> x0 >> y0;
   cout << "Enter second point: ";
   cin >> x1 >> y1;

   glClear(GL_COLOR_BUFFER_BIT);
   drawWindow(XWMIN, YWMIN, XWMAX, YWMAX);
   glColor3f(1, 0, 0);
   drawLine(x0, y0, x1, y1);
   int xc1, yc1, xc2, yc2;
   clipLine(xc1, yc1, xc2, yc2, x0, y0, x1, y1);
   glColor3f(0, 0, 1);
   if(xc1 != -1 && yc1 != -1 && xc2 != -1 && yc2 != -1) drawLine(xc1, yc1, xc2, yc2);
  cout << xc1 << " " << yc1 << "  " << xc2 << " " << yc2 << endl;
}

int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

```
    glutInitWindowPosition(100, 200);
    glutInitWindowSize(800, 600);
    glutCreateWindow("Mid point subdivision");
    glClearColor(1, 1, 1, 0);
    glColor3f(0,0,0);
    gluOrtho2D(0, 800, 0, 600);
    glutDisplayFunc(mpsd);
    glutMainLoop();
}
```

Output :

```
Enter first point: 100 90
Enter second point: 97 68
-1 -1   -1 -1
Enter first point: 45 78
Enter second point: 23 75
-1 -1   -1 -1
Enter first point: 93 35
```