Name of the Institute ..... RVCE .....

Laboratory Certificate

This is to certify that Smt./ ~~Sri~~ ..... N NiKhiL .....

has satisfactorily complete the course of experiments in

Practical ..... C Gr Lab ..... *presribed by*

the .....

in the Laboratory of this College in the year 2020-2021

Signature of the Teacher in charge of the batch

Head of the Department

Name of the Candidate ..... N. NiKhiL .....

..... Reg. No. 1RV18CS414 .....

Examination Centre .....

Date of Practical Examination .....

Name of Experiment..... Date.....

Experiment No..... Experiment Result..... Page No 01

PROGRAM - 01

write a Program to generate a line using Bresenham's line drawing technique. Consider slopes greater than one and slopes less than one. User must able to draw as many lines and specify inputs through Keyboard/ mouse.

```
#include <iostream>
#include <GL/glut.h>
#include<time.h>
#include<stdio.h>
using namespace std;
int X1, X2, YC1, Y2;
int flag=0;

void draw_pixel(int x, int y)
{
    glColor3f(1,0,0);
    glBegin(GL_POINTS);
    glVertex2i(x,y);
    glEnd();
    glFlush();
}

void draw_line()
{
   int dx, dy, i, e;
   int incx, incy, inc1, inc2;
   int x, y;
```

```
dx = x2 - x1;
dy = y2 - yc1;
if (dx < 0)
    dx = -dx;
if (dy < 0)
    dy = -dy;
incx = 1;
if (x2 < x1)
   incx = -1;
incy = 1;
if (y2 < yc1)
   incy = -1;
x = x1;
y = yc1;
if (dx > dy)
{
   draw_pixel(x, y);
   e = 2*dy - dx;
   inc1 = 2*(dy - dx);
   inc2 = 2*dy;
   for (i=0; i<dx; i++)
   { if (e > 0)
     {  y += incy;
        e += inc1;
     }
     else
        e += inc2;
    x += incx;
    draw_pixel(x, y);
   }
}
else
{
```

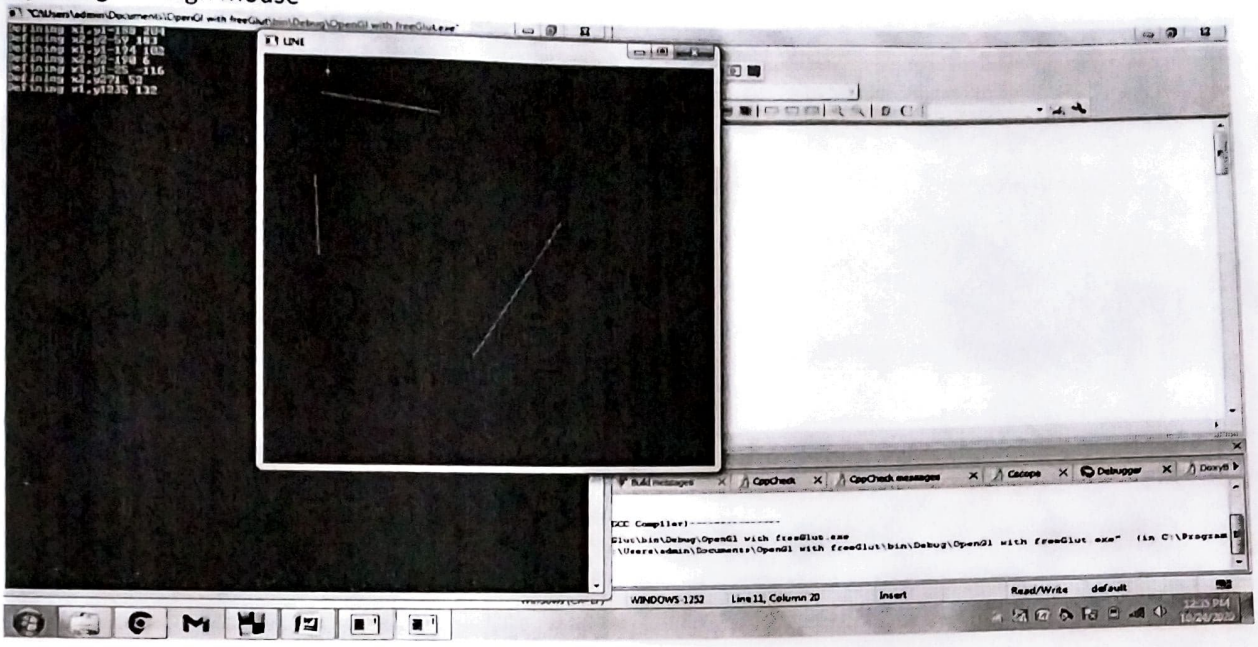
```
    draw_pixel(x,y);
    e = 2*dx - dy;
    inc1 = 2*(dx-dy);
    inc2 = 2*dx;
    for(i=0; i<dy; i++)
    {   if(e>0)
        {   x+=incx;
            e+=inc1;
        }
        else
            e+=inc2;
        y+=incy;
        draw_pixel(x,y);
    }
}
glFlush();
}

void myinit()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glClearColor(1,1,1,1);
    gluOrtho2D(-250,250,-250,250);
}

void mymouse(int button, int state, int x, int y)
{
    switch(button)
    {
    case GLUT_LEFT_BUTTON;
        if(state==GLUT_DOWN)
        {
```

```
        if(flag==0)
        {
            Printf("Defining x1,y1");
            x1 = x-250;
            yc1 = 250-y;
            flag++;
            cout<<x1<<" "<<yc1<<"\n";
        }
        else
        {
            Printf("Defining x2,y2");
            x2 = x-250;
            y2 = 250-y;
            flag=0;
            cout<<x2<<" "<<y2<<"\n";
            draw_line();
        }
    }
    break;
}
}

void display(){ }

int main(int ac, char* av[])
{
    /*
    //For Keyboard
    cout<<"x1\n";
    cin>>x1;
    cout<<"y1\n";
    cin>>yc1;
    cout<<"x2\n";
    cin>>x2;
    cout<<"y2\n";
    cin>>y2;
    //End of Keyboard
    */
```

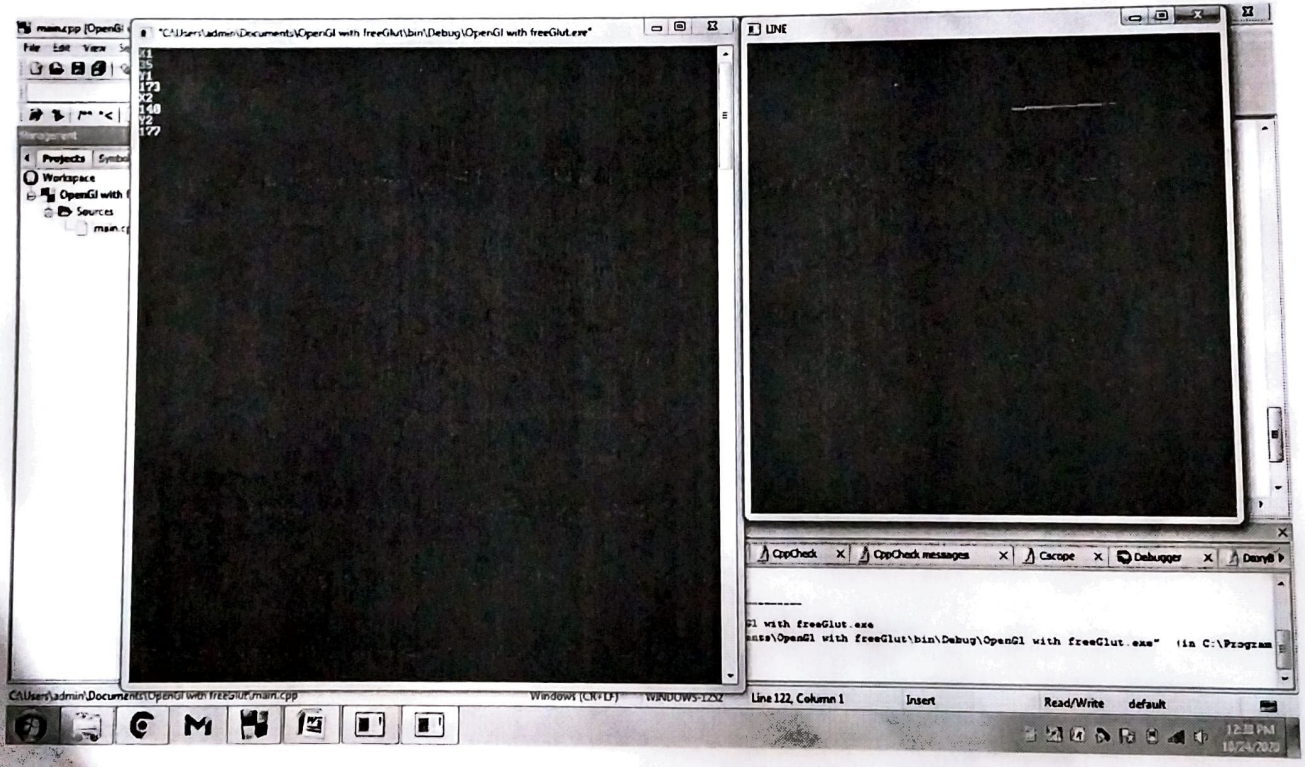
```
glutInit(&ac, av);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(100, 200);
glutCreateWindow("LINE");
myinit();
glutMouseFunc(myMouse); //include to mouse, remove while
                          using keyboard
//draw_line(); //include to use keyboard, remove for mouse
glutDisplayFunc(display);
glutMainLoop();
}
```

OUTPUT :

1. Inputting Through Mouse



2. Inputting Through Keyboard



PROGRAM - 02

Write a program to generate a circle and ellipse using Bresenham's circle drawing and ellipse drawing techniques. Use two windows to draw circle in one window and ellipse in the other window. User can specify inputs through keyboard/mouse.

```
#include<gl/glut.h>
#include<stdio.h>
#include<math.h>
int xc, yc, r;
int rx, ry, xcc, ycc;
void draw_circle(int xc, int yc, int x, int y)
{   glBegin(GL_POINTS);
    glVertex2i(xc+x, yc+y);
    glVertex2i(xc-x, yc+y);
    glVertex2i(xc+x, yc-y);
    glVertex2i(xc-x, yc-y);
    glVertex2i(xc+y, yc+x);
    glVertex2i(xc-y, yc+x);
    glVertex2i(xc+y, yc-x);
    glVertex2i(xc-y, yc-x);
    glEnd();
}

void circlebres()
{
    glClear(GL_COLOR_BUFFER_BIT);
```

```
int x=0, y=r;
int d = 3-2*r;
while(x<=y)
{ draw_circle(xc, yc, x, y);
  x++;
  if(d<0)
     d=d+4*x+6;
  else
  { y--;
    d=d+4*(x-y)+10;
  }
  draw_circle(xc, yc, x, y);
}
glFlush();
}

int P1_x, P2_x, P1_y, P2_y;
int point1_done=0;

void myMousefunccircle(int button, int state, int x, int y)
{
  if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
      point1_done==0)
  {
      P1_x = x - 250;
      P1_y = 250 - y;
      point1_done = 1;
  }
  else if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
  {
```

```
        P2_x = x-250;
        P2_y = 250-y;
        xc = P1_x;
        yc = P1_y;
        float exp=(P2_x-P1_x)*(P2_x-P1_x)+(P2_y-P1_y)*
                                    (P2_y-P1_y);
        r=(int)(sqrt(exp));
        circlebres();
        point1_done=0;
    }
}

//Ellipse drawing

void draw_ellipse(int xce, int yce, int x, int y)
{
    glBegin(GL_POINTS);
    glVertex2i(x+xce, y+yce);
    glVertex2i(-x+xce, y+yce);
    glVertex2i(x+xce, -y+yce);
    glVertex2i(-x+xce, -y+yce);
    glEnd();
}

void midptellipse()
{
    glClear(GL_COLOR_BUFFER_BIT);
    float dx, dy, d1, d2, x, y;
    x=0;
    y=ry;
```

```
d1 = (ry*ry) - (rx*rx*ry) + (0.25*rx*rx);
dx = 2*ry*ry*x;
dy = 2*rx*rx*y;

while (dx<dy)
{   draw_ellipse(xce, yce, x, y);
    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)
{
    draw_ellipse(xce, yce, x, y);
    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);
    }
  }
  glFlush();
}
int P1e_x, P2e_x, P1e_y, P3e_x, P3e_y;
int Point1e_done = 0;

void myMouseFunc(int button, int state, int x, int y)
{
  if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
    Point1e_done == 0)
  {   P1e_x = x - 250;
      P1e_y = 250 - y;
      xce = P1e_x;
      yce = P1e_y;
      Point1e_done = 1;
  }
  else if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
          Point1e_done == 1)
  {
```

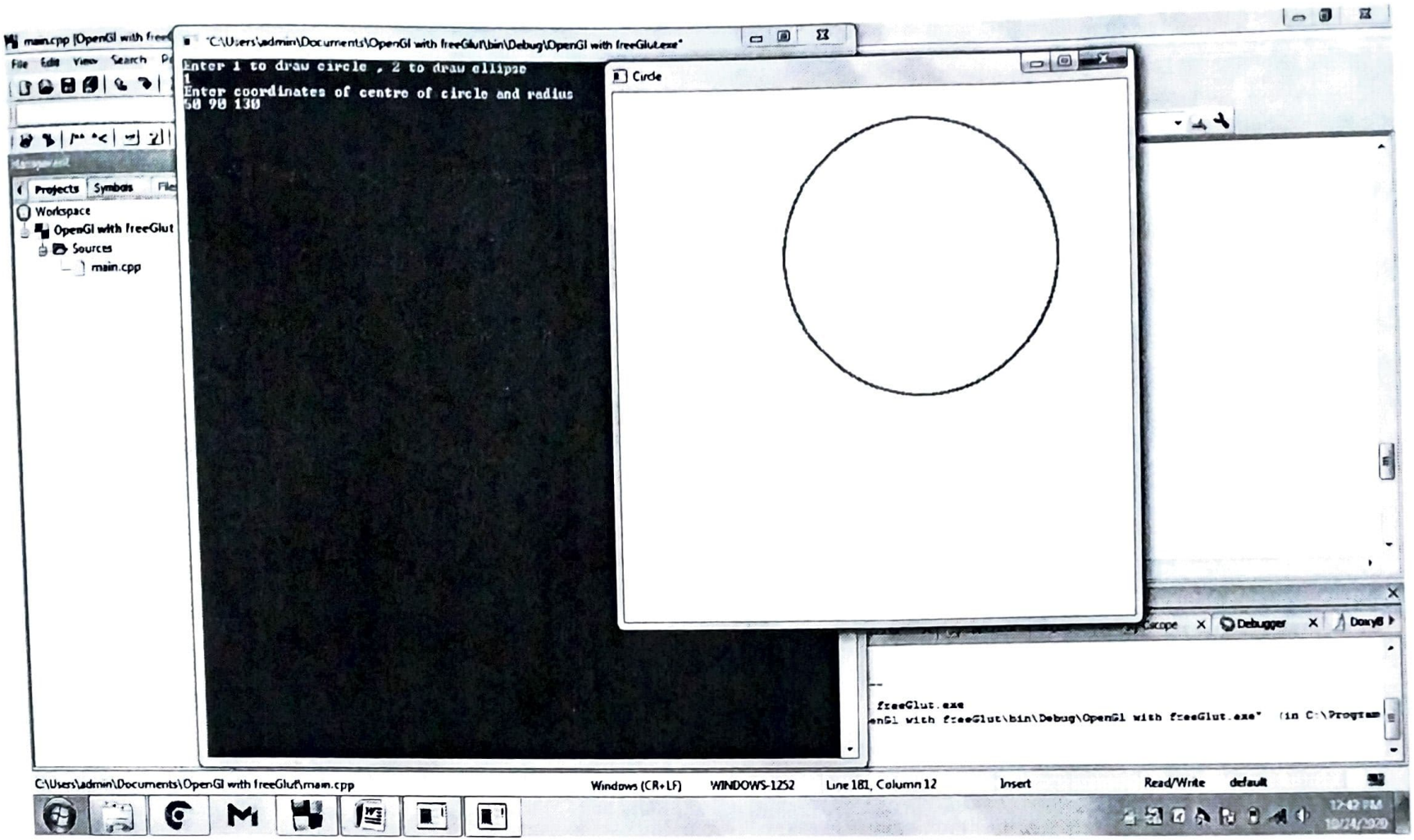
```
        p2e_x = x-250;
        p2e_y = 250-y;
        float exp = (p2e_x - p1e_x) * (p2e_x - p1e_x)+(p2e_y - p1e_y)
                    * (p2e_y - p1e_y);
        rx = (int)(sqrt(exp));
        point1e_done = 2;
    }
    else if ( button == GLUT_LEFT_BUTTON && state == GLUT_DOWN
        && point1e_done == 2 )
    {
        p3e_x = x - 250;
        p3e_y = 250 - y;
        float exp = (p3e_x - p1e_x) * (p3e_x - p1e_x)+(p3e_y - p1e_y)
                    * (p3e_y - p1e_y);
        ry = (int)(sqrt(exp));
        midptellipse();
        point1e_done = 0;
    }
}
void myDrawing()
{}
void myDrawing()
{}

void minit(){
        glClearColor(1, 1, 1, 1);
        glColor3f(1.0, 0.0, 0.0);
        glPointSize(3.0);
        gluOrtho2D(-250, 250, -250, 250);
}
```

```
int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0,0);
    // For mouse
    int id1 = glutCreateWindow("circle");
    glutSetWindow(id1);
    glutMouseFunc(myMouseFuncCircle);
    glutDisplayFunc(myDrawingC);
    minit();
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(600,100);
    int id2 = glutCreateWindow("Ellipse");
    glutSetWindow(id2);
    glutMouseFunc(myMouseFunc);
    glutDisplayFunc(myDrawing);
    // End mouse

    // For keyboard
    printf("Enter 1 to draw circle, 2 to draw ellipse\n");
    int ch;
    scanf("%d", &ch);
    switch (ch) {
    case 1:
    printf("Enter coordinates of centre of circle & radius\n");
    scanf("%d%d%d", &xc, &yc, &r);
    glutCreateWindow("circle");
```

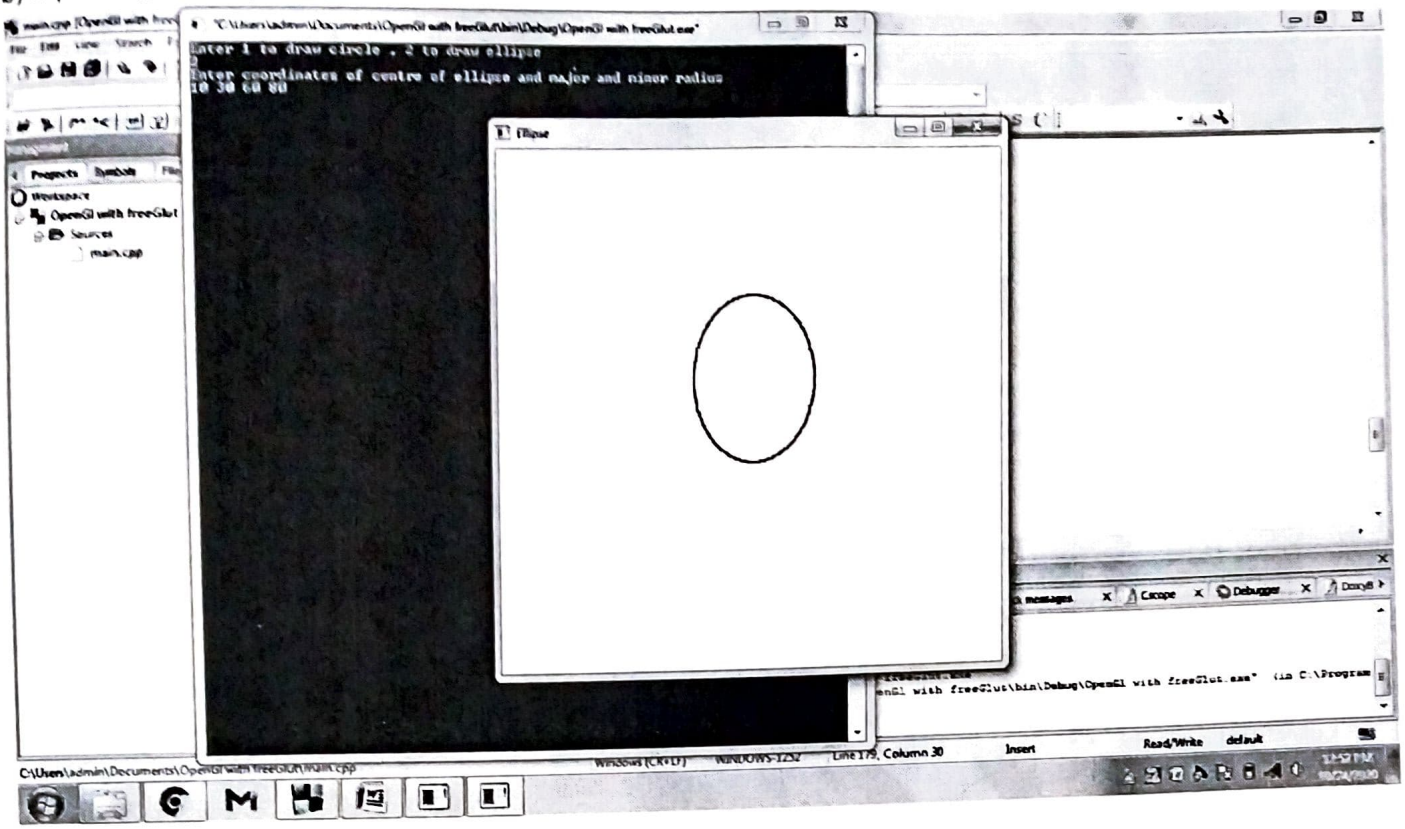
```
glutDisplayFunc(circlebres);
break;
case 2:
printf("Enter coordinates of centre of ellipse & major
and minor radius \n");
scanf("%d %d %d %d", &xce, &yce, &rx, &ry);
glutCreateWindow("Ellipse");
glutDisplayFunc(midPtEllipse);
break;
}
//End of keyboard
init();
glutMainLoop();
}
```

OUTPUT :

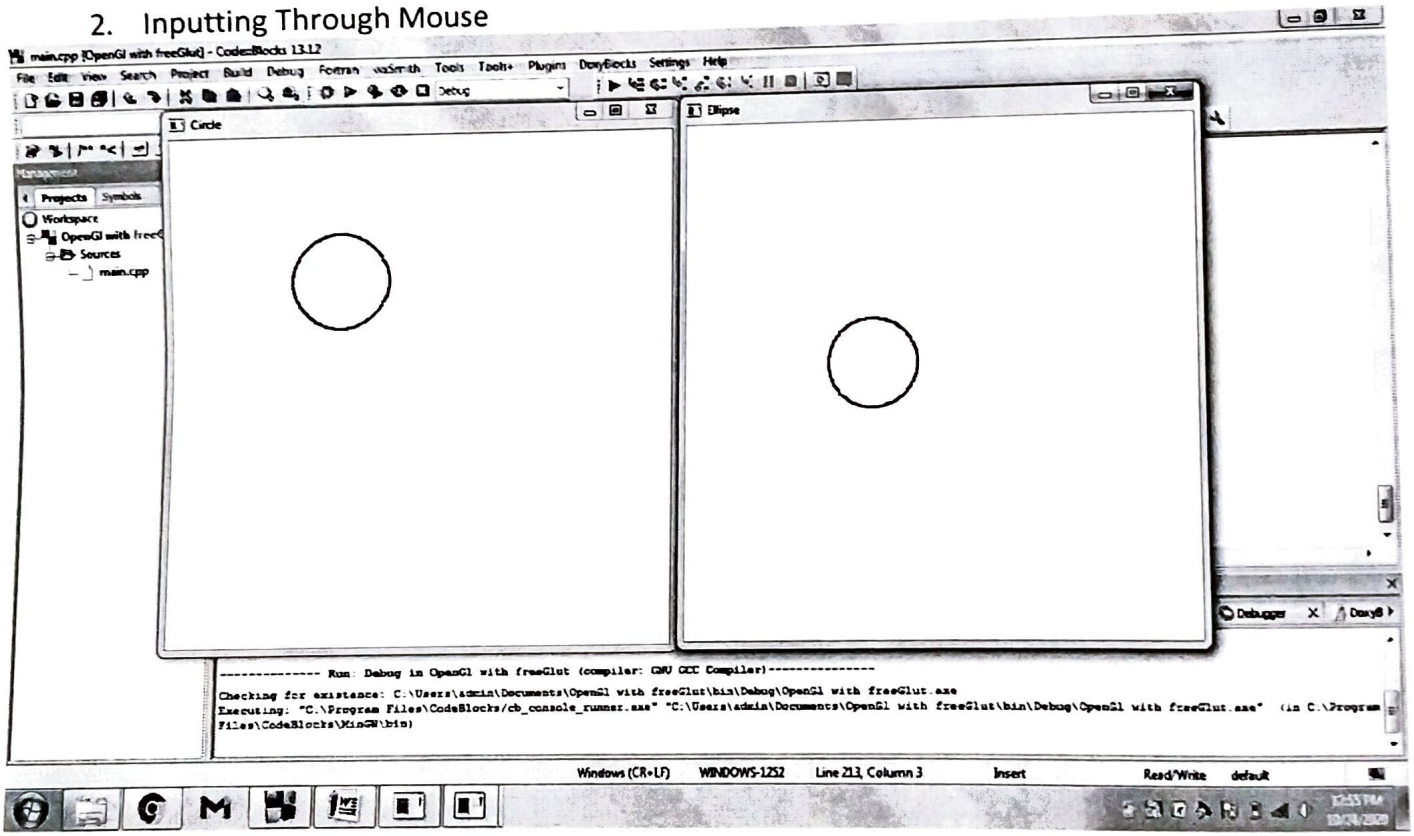
1. a) Inputting Through Key board



b) Inputting Through keyboard



2. Inputting Through Mouse



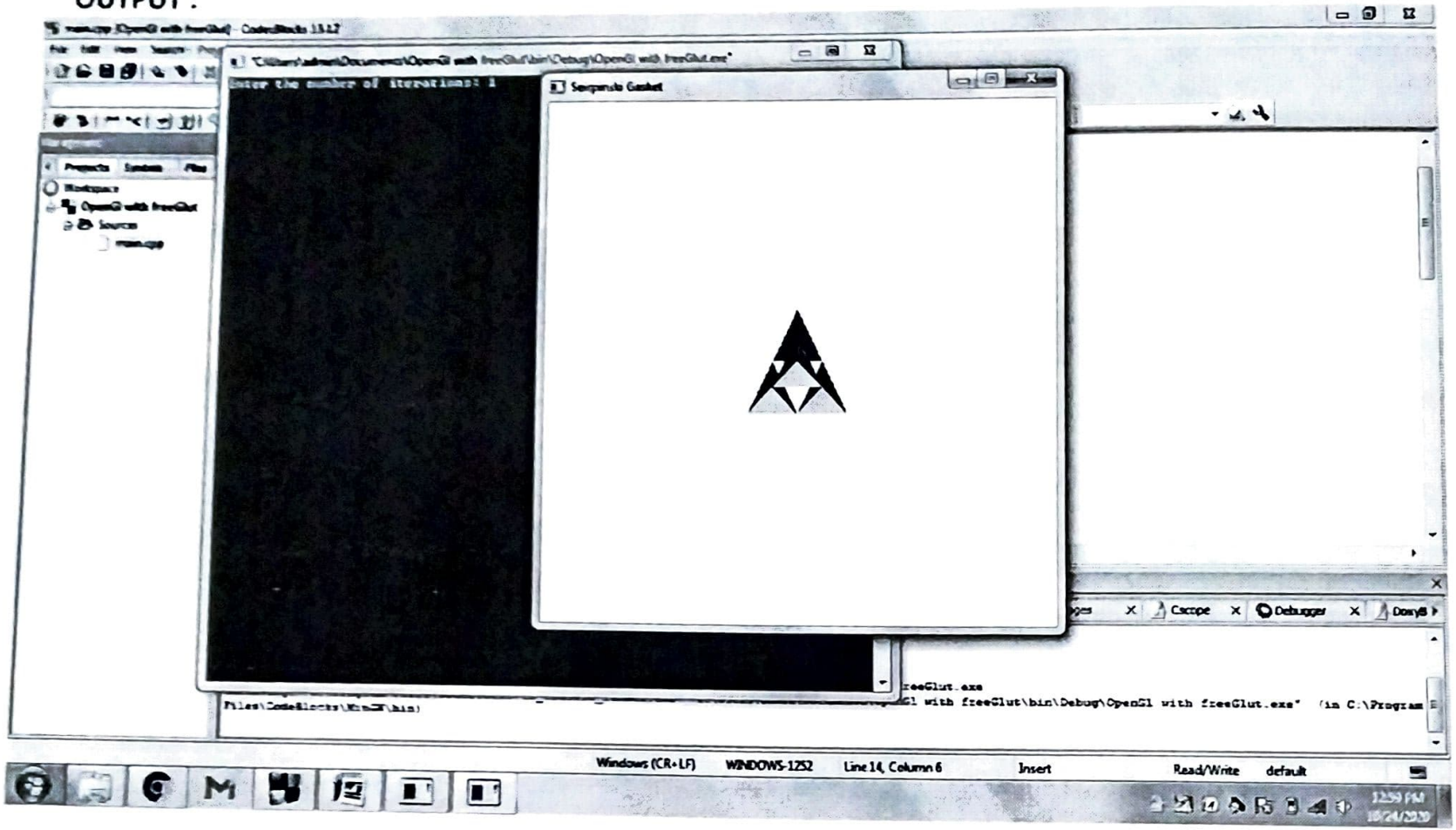
PROGRAM - 03

Write a program to recursively subdivide a tetrahedron to form 3D sierpinski gasket. The number of recursive steps to be specified at execution time.

```
#include<gl/glut.h>
#include<stdio.h>
int m;
typedef float point[3];
point tetra[4] = {{0,100,-100}, {0,0,100}, {100,-100,-100}, {100,-100,100}};
void tetrahedron(void);
void myinit(void);
void divide_triangle(point a, point b, point c, int m);
void draw_triangle(point p1, point p2, point p3);

int main(int argv, char** argc)
{   printf("Enter the number of iterations:");
    scanf("%d", &m);
    glutInit(&argv, argc);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowPosition(100,200);
    glutInitWindowSize(500,500);
    glutCreateWindow("sierpinski Gasket");
    glutDisplayFunc(tetrahedron);
    glEnable(GL_DEPTH_TEST);
    myinit();
    glutMainLoop();
}
```

```
void divide_triangle(point a, point b, point c, int m)
{
    point v1, v2, v3;
    int j,
    if(m>0){
        for(j=0; j<3; j++)
            v1[j] = (a[j]+b[j])/2;
        for(j=0; j<3; j++)
            v2[j] = (a[j]+c[j])/2;
        for(j=0; j<3; j++)
            v3[j] = (b[j]+c[j])/2;
        divide_triangle (a, v1, v2, m-1);
        divide_triangle (c, v2, v3, m-1);
        divide_triangle (b, v3, v1, m-1); }
    else
        draw_triangle(a, b, c); }

void myinit(){
    glClearColor(1,1,1,1);
    glOrtho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0); }

void draw_triangle (point p1, point p2, point p3)
{
    glBegin(GL_TRIANGLES);
    glVertex3fv(p1);
    glVertex3fv(p2);
    glVertex3fv(p3);
    glEnd();
}
```

OUTPUT :


Enter the number of iterations: 1
Sierpinski Gasket
Windows (CR+LF)
WINDOWS-1252
Line 14, Column 6
Insert
Read/Write
default

Name of Experiment | Date | Page No. 16

Experiment No. | Experiment Result

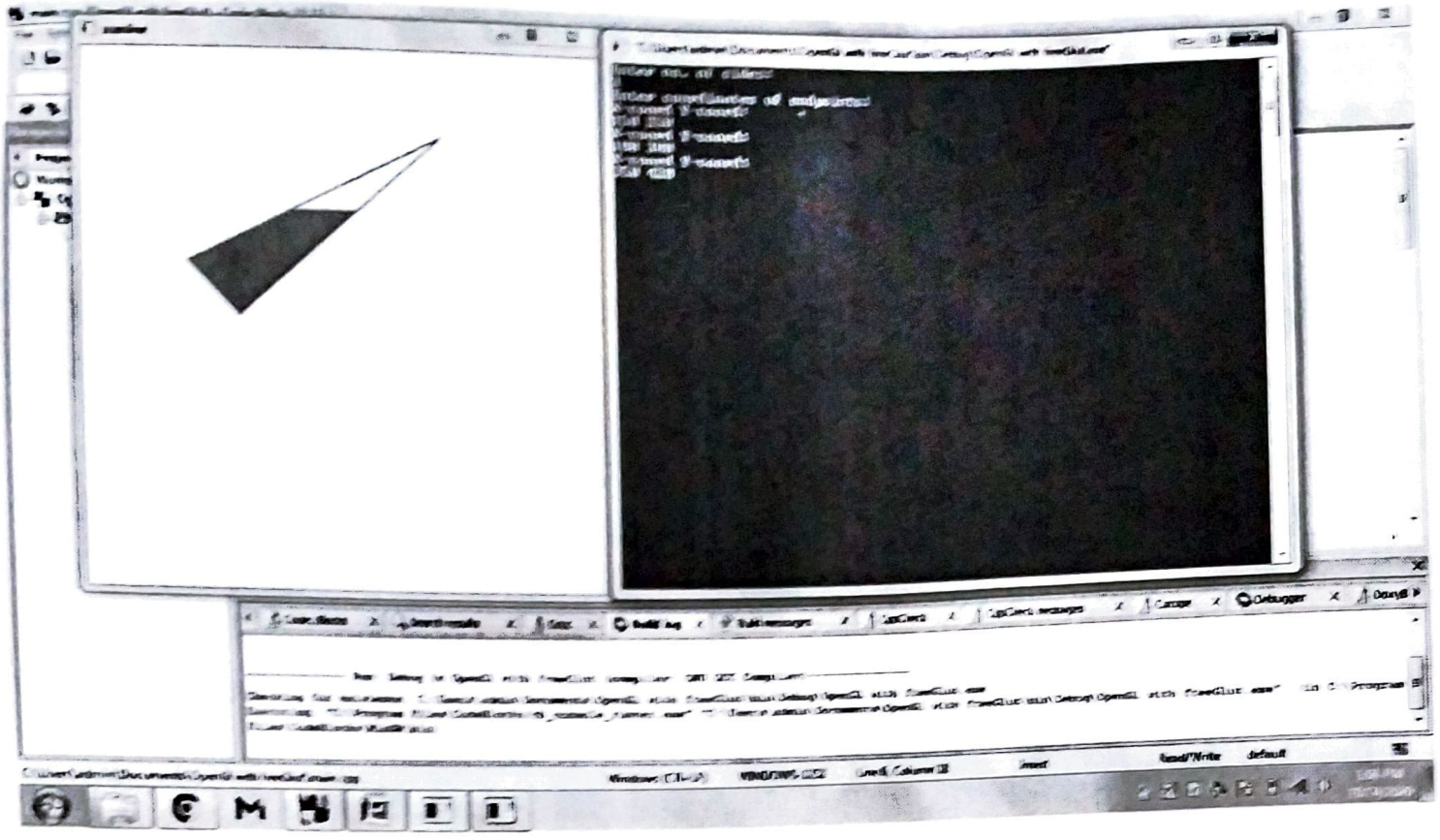
PROGRAM - 04

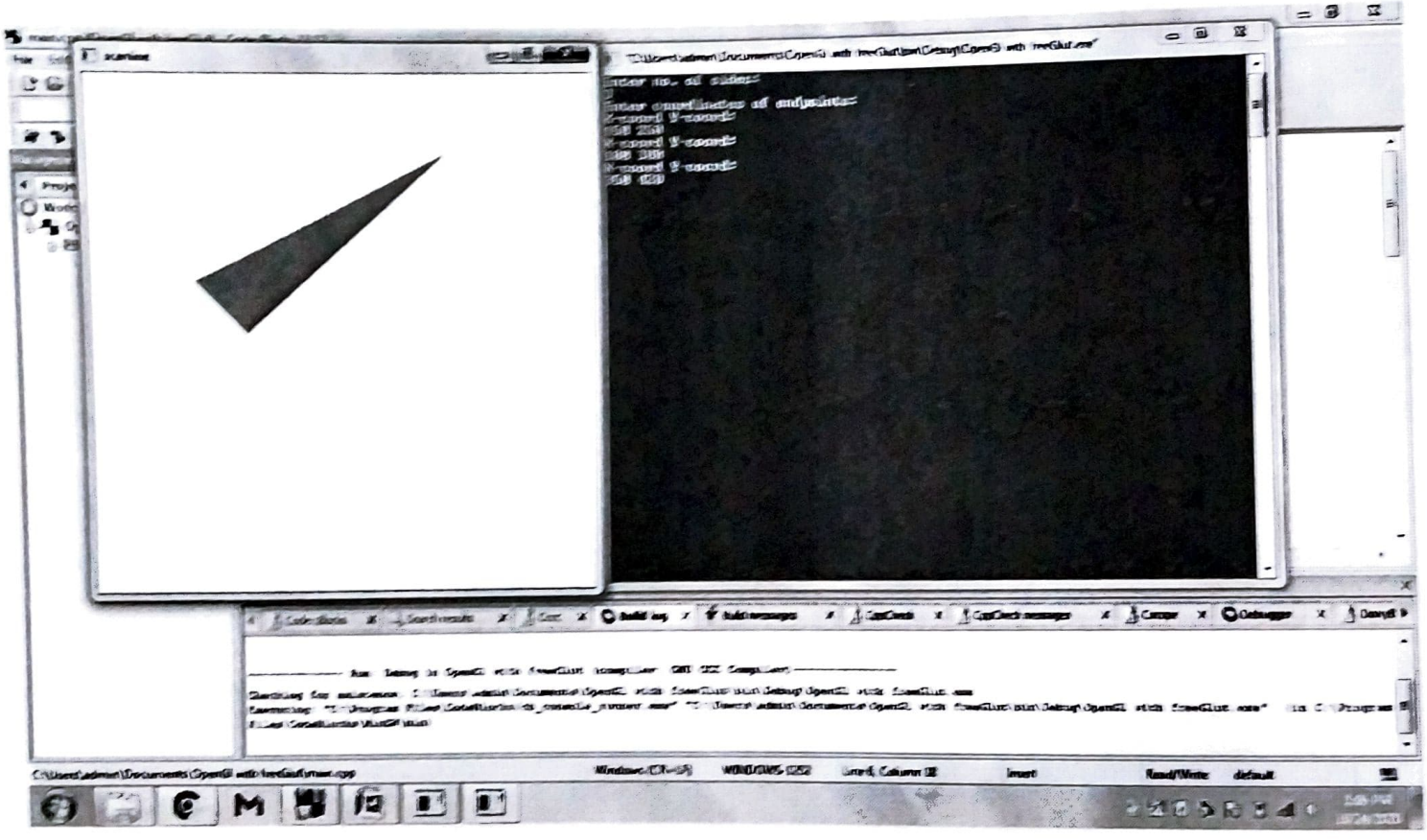
Write a program to fill any given polygon using scan-line area filling algorithm

```
#include<stdlib.h>
#include<gl/glut.h>
#include<algorithm>
#include<iostream>
#include<windows.h>
#include<stdio.h>
using namespace std;
float x[100], y[100];
int n, m;
int wx=500, wy=500;
static float intx[10]={0};
void draw_line(float x1, float y1,
               float x2, float y2){
    Sleep(100);
    glColor3f(1,0,0);
    glBegin(GL_LINES);
    glVertex2f(x1,y1);
    glVertex2f(x2,y2);
    glEnd();
    glFlush(); }

void edgeDetect(float x1, float y1, float x2, float y2, int scanline){
    float temp;
    if(y2<y1){
        temp=x1; x1=x2; x2=temp;
        temp=y1; y1=y2; y2=temp;
    }
    if(scanline>y1 && scanline<y2)
        intx[m++]=x1+(scanline-y1)*
                  (x2-x1)/(y2-y1);
}
void myInit(){
    glClearColor(1,1,1,1);
    glColor3f(0,0,1);
    glPointSize(1);
    glOrtho2D(0,wx,0,wy); }

void display_filled_polygon(){
    glClear(GL_COLOR_BUFFER_BIT);
    glLineWidth(2);
    glBegin(GL_LINE_LOOP);
    for(int i=0; i<n; i++)
        glVertex2f(x[i], y[i]);
    glEnd();
    scanfill(x,y); }

void edgeDetect(float x1, float y1, float x2, float y2, int scanline){
    float temp;
    if(y2<y1){
```

```
void scanfill(float x[], float y[]){
    for(int s1=0; s1<=wy; s1++){
        m=0;
        for(int i=0; i<n; i++){
            edgeDetect(x[i], y[i], x[(i+1)%n], y[(i+1)%n], s1); }
    sort(int x, (int x+m));
    if(m>=2)
        for(int i=0; i<m; i=i+2)
            drawline(int x[i], s1, int x[i+1], s1);
}
void main(int ac, char* av[]){
    glutInit(&ac, av);
    Printf("Enter no of sides:\n");
    scanf("%d", &n);
    Printf(" Enter coordinates of endpoints:\n");
    for(int i=0; i<n; i++)
    {   printf("X-coord Y-coord:\n");
        scanf("%f %f", &x[i], &y[i]);
    }
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("scanline");
    glutDisplayFunc(display_filled_polygon);
    myInit();
    glutMainLoop();
}
```

OUTPUT :





Name of Experiment | Date | Page No. 18

Experiment No. | Experiment Result

PROGRAM - 05

Write a program to create a house like a figure and perform the following operations

i) Rotate it about a given fixed point using openGL transformation functions

ii) Reflect it about an axis $y = mx + c$ using openGL transformation functions.

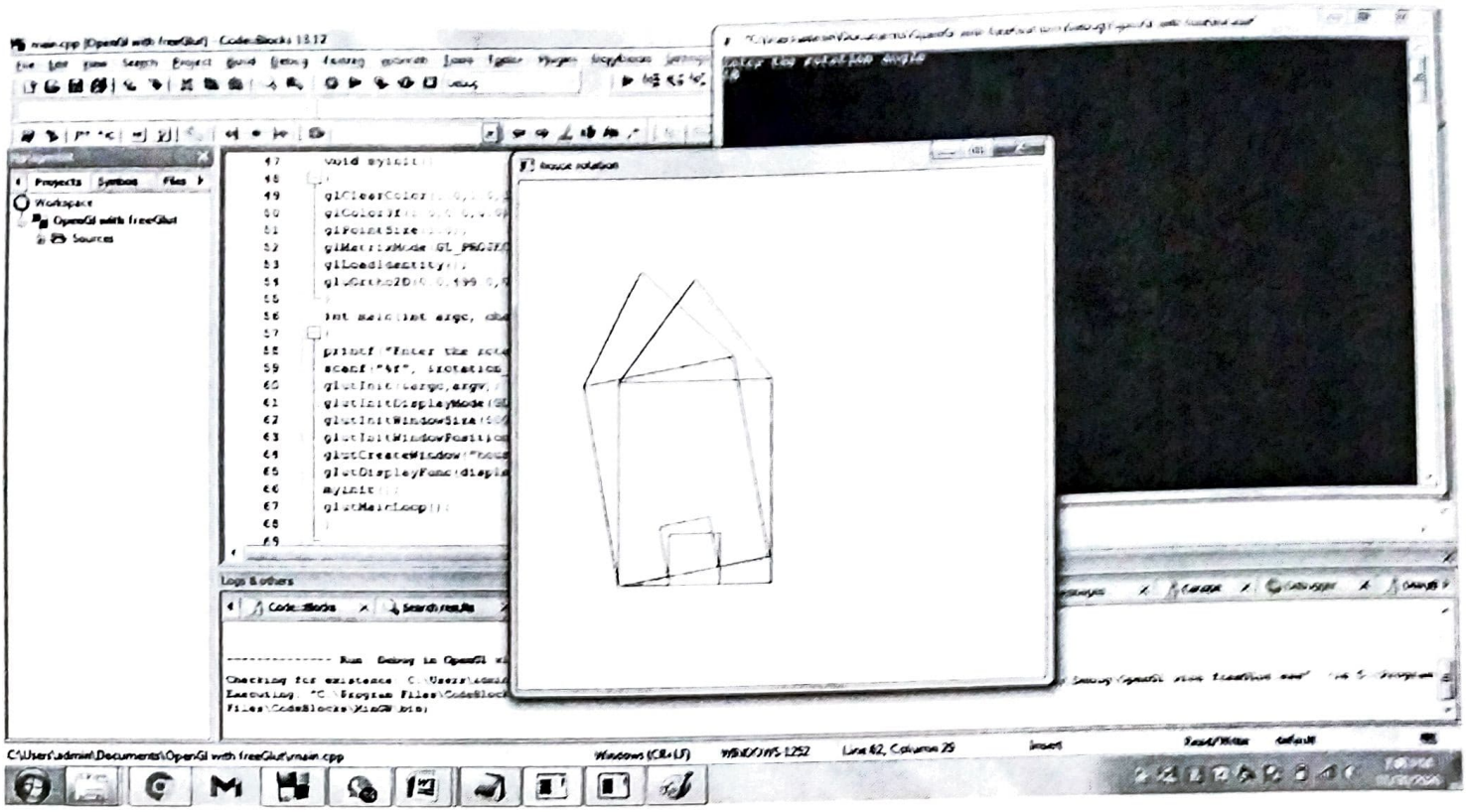
```
#include<gl/glut.h>
#include<math.h>
#include<stdio.h>
float house[11][2]={{100,200},{200,250},{300,200},{100,200},{100,100},
{175,100},{175,150},{225,150},{225,100},{300,100},{300,200}};
int angle;
float m,c,theta;
void display()
{ glClearColor(1,1,1,0);
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-450,450,-450,450);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  //normal house
  glColor3f(1,0,0);
  glBegin(GL_LINE_LOOP);
  for(int i=0;i<11;i++)
    glVertex2fv(house[i]);
  glEnd();
  glFlush();
  //Rotated house
  glPushMatrix();
  glTranslatef(100,100,0);
  glRotatef(angle,0,0,1);
  glTranslatef(-100,-100,0);
  glColor3f(1,1,0);
  glBegin(GL_LINE_LOOP);
  for(int i=0;i<11;i++)
    glVertex2fv(house[i]);
  glEnd();
  glPopMatrix();
  glFlush();
```

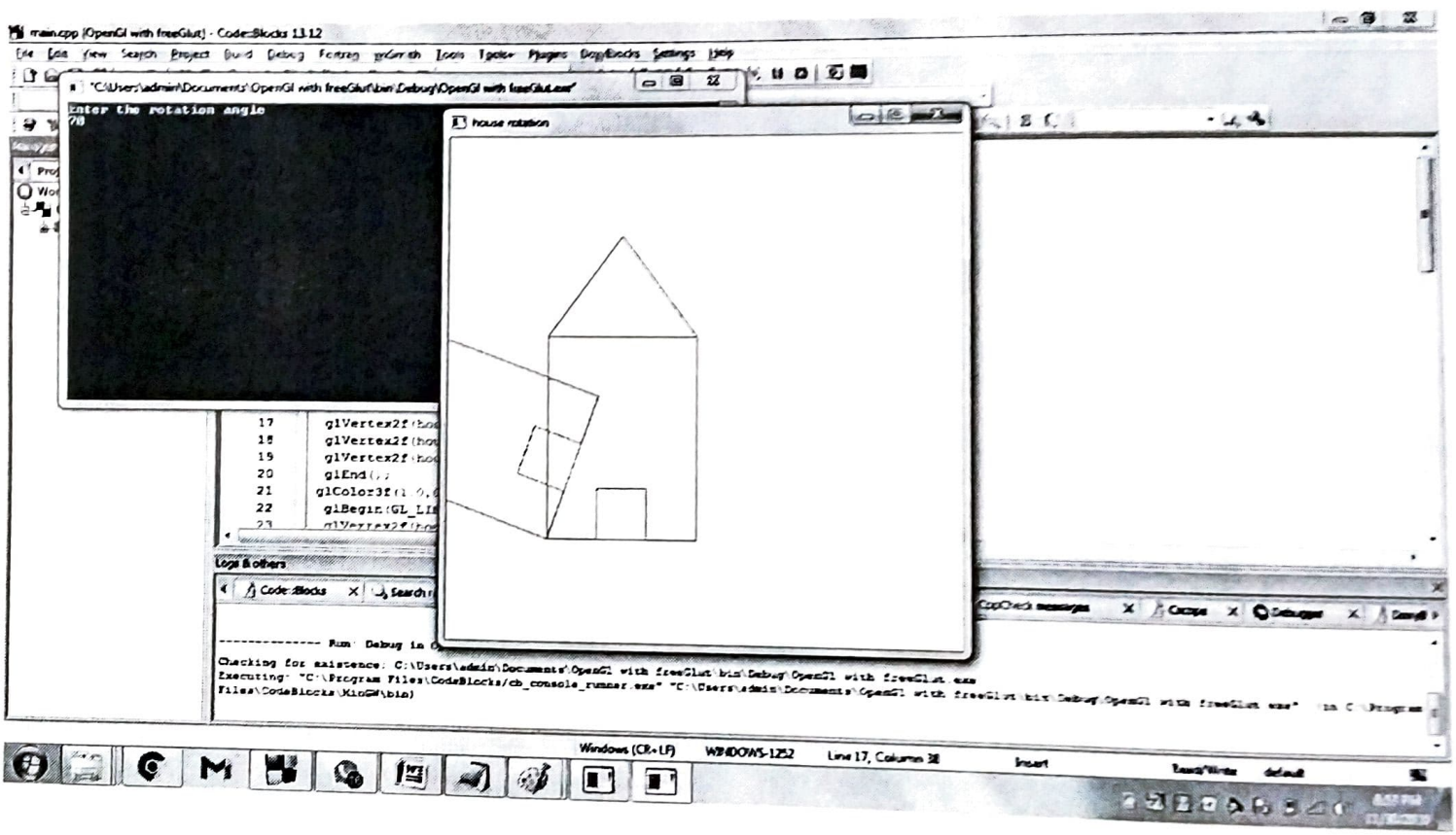
```
void display2(){
glClearColor(1,1,1,0);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-450,450,-450,450);
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
//normal house
glColor3f(1,0,0);
glBegin(GL_LINE_LOOP);
for(int i=0; i<11; i++)
  glVertex2fv(house[i]);
glEnd();
glFlush();
//line
float x1=0, x2=500;
float y1=m*x1+c;
float y2=m*x2+c;
glColor3f(1,1,0);
glBegin(GL_LINES);
glVertex2f(x1,y1);
glVertex2f(x2,y2);
glEnd();
glFlush();
//Reflected
glPushMatrix();
glTranslate(0,c,0);
theta=atan(m);
theta=theta*180/3.14;
glRotatef(theta,0,0,1);
glScalef(1,-1,1);
glRotatef(-theta,0,0,1);
glTranslatef(0,-c,0);
glBegin(GL_LINE_LOOP);
for(int i=0; i<11; i++)
  glVertex2fv(house[i]);
glEnd();
glPopMatrix();
glFlush();
}
void myInit(){
  glClearColor(1.0,1.0,1.0,1.0);
  glColor3f(1.0,0.0,0.0);
  glLineWidth(2.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(-450,450,-450,450);
}
void mouse(int btn, int state, int x, int y){
  if(btn==GLUT_LEFT_BUTTON && state==
  GLUT_DOWN){ display(); }
  else if(btn==GLUT_RIGHT_BUTTON &&
  state==GLUT_DOWN){
    display2();
  }
}
```

chandra's

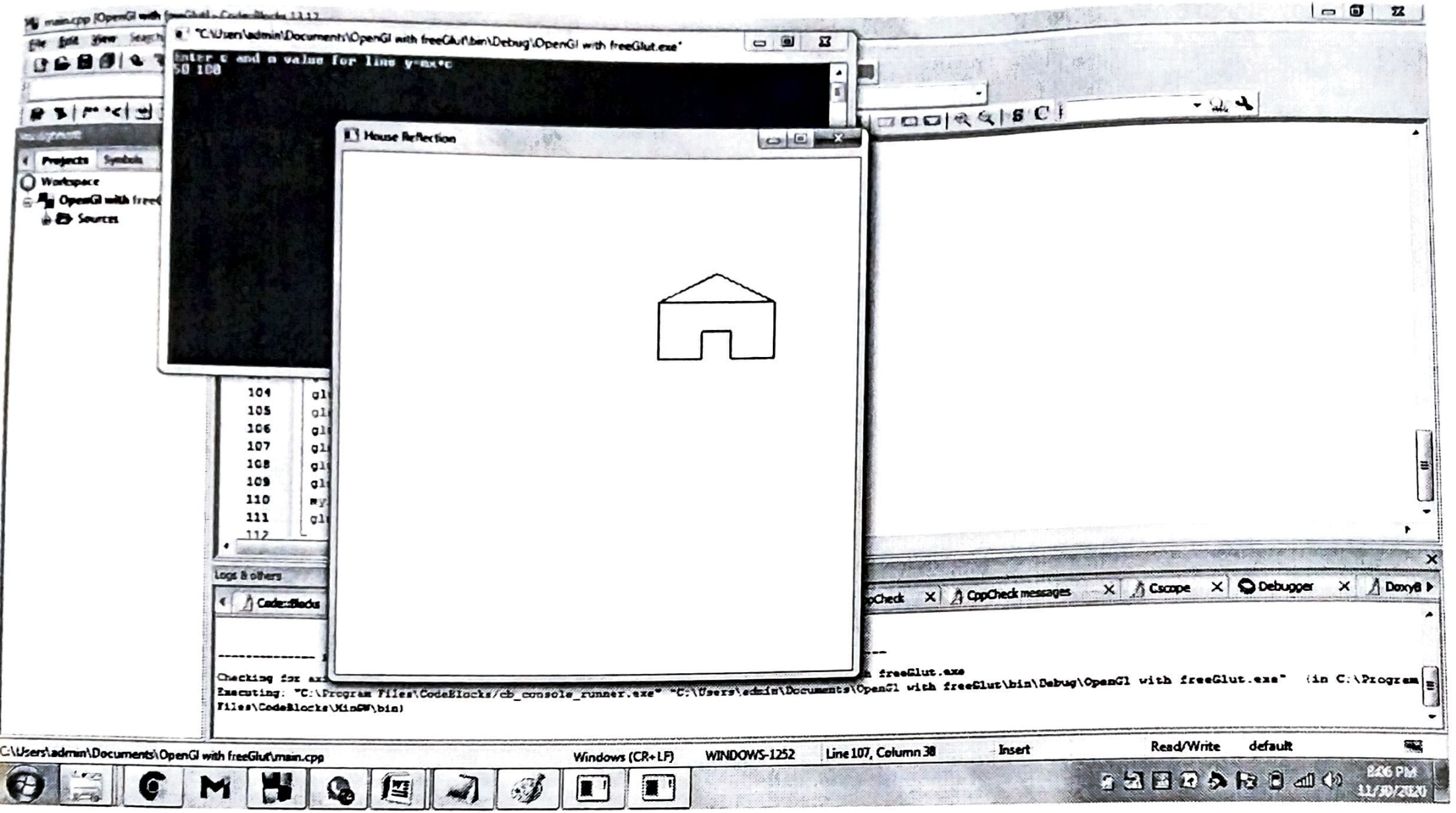
```
void main(int argc, char** argv)
{
    printf("Enter the rotation angle");
    scanf("%d", &angle);
    printf("Enter c and m value for line y=mx+c \n");
    scanf("%f %f", &c, &m);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(900, 900);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("house rotation");
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    myInit();
    glutMainLoop(); }
```

OUTPUT:





OUTPUT:


C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe
Enter c and a value for line y=mx+c
50 100
House Reflection
Logs & others
CppCheck messages
Cscope
Debugger
Doxyg
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe"
Files\CodeBlocks\MinGW\bin)
C:\Users\admin\Documents\OpenGl with freeGlut\main.cpp
Windows (CR+LF)
WINDOWS-1252
Line 107, Column 38
Insert
Read/Write
default

Name of Experiment..... Date.....

Experiment No..... Experiment Result..... Page No. 21

PROGRAM-6

Write a Program to implement Cohen-Sutherland line clipping algorithm. Make a provision to specify the input of multiple lines, window for clipping & viewport for displaying the clipped image.

```
#include<stdio.h>
#include<GL/glut.h>
double xmin=50, ymin=50,          // window boundaries
       xmax=100, ymax=100;
double xvmin=200, yvmin=200,      // viewport boundaries
       xvmax=300, yvmax=300;
const int RIGHT=8;                // bit codes for right, left, top, bottom
const int LEFT =2;
const int TOP = 4;
const int BOTTOM=1;
OutCode ComputeOutCode(double x,  // to compute bit codes of point
        double y);
void CohenSutherland(double x0, double y0, double x1, double y1)
{  OutCode outcode0, outcode1, outcodeOut; // outcode for P0 & P1
   bool accept = false, done=false;        // point lies outside the clip rectangle.
   outcode0 = ComputeOutCode(x0,y0);  // compute outcode
   outcode1 = ComputeOutCode(x1,y1);
   do{
     if(!(outcode0 | outcode1)) // logical or is 0 trivially accept & exit
     {  accept=true;
     }  done = true;
     else if (outcode0 & outcode1) // logical or is not 0. Trivially reject & exit
        done=true;
     else{
```

chandra's

```
//failed both tests, so calculate the line segment to clip from an
outside point to an intersection with clip edge
double x,y;
//At least one endpoint is outside the clip rectangle; pick it.
//now find the intersection point
if (outcodeOut & TOP) //point is above clip rectangle
{
   x = x0 + (x1 - x0) * (ymax - y0)/(y1 - y0);
   y = ymax;
}
else if (outcodeOut & BOTTOM) //point is below the clip rectangle
   x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
   y = ymin;
}
else if (outcodeOut & RIGHT) //point is to the right of clip rectangle
{  y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
   x = xmax; }
else
{  y0 = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
   x = xmin; }

//now we move outside point to intersection point to clip
if (outcodeOut == outcode0)
{  x0 = x;
   y0 = y;
   outcode0 = computeOutCode(x0, y0);
}
else
{  x1 = x;
   y1 = y;
   outcode1 = computeOutCode(x1, y1);
}
}
```

```
} while (!done);
if(accept)
{ //window to viewport mapping
  double sx = (xvmax-xvmin)/(xmax-xmin); //scale parameters
  double sy = (yvmax-yvmin)/(ymax-ymin);
  double vx0 = xvmin+(x0-xmin)*sx;
  double vy0 = yvmin+(y0-ymin)*sy;
  double vx1 = xvmin+(x1-xmin)*sx;
  double vy1 = yvmin+(y1-ymin)*sy;
  glColor3f(1.0,0.0,0.0); // draw a red colored viewport
  glBegin(GL_LINE_LOOP);
  glVertex2f(xvmin, yvmin);
  glVertex2f(xvmax, yvmax);
  glVertex2f(xvmax, yvmax);
  glVertex2f(xvmin, yvmax);
  glEnd();
  glColor3f(0.0,0.0,1.0);
  glBegin(GL_LINES);
  glVertex2d(vx0, vy0);
  glVertex2d(vx1, vy1);
  glEnd();
}
```

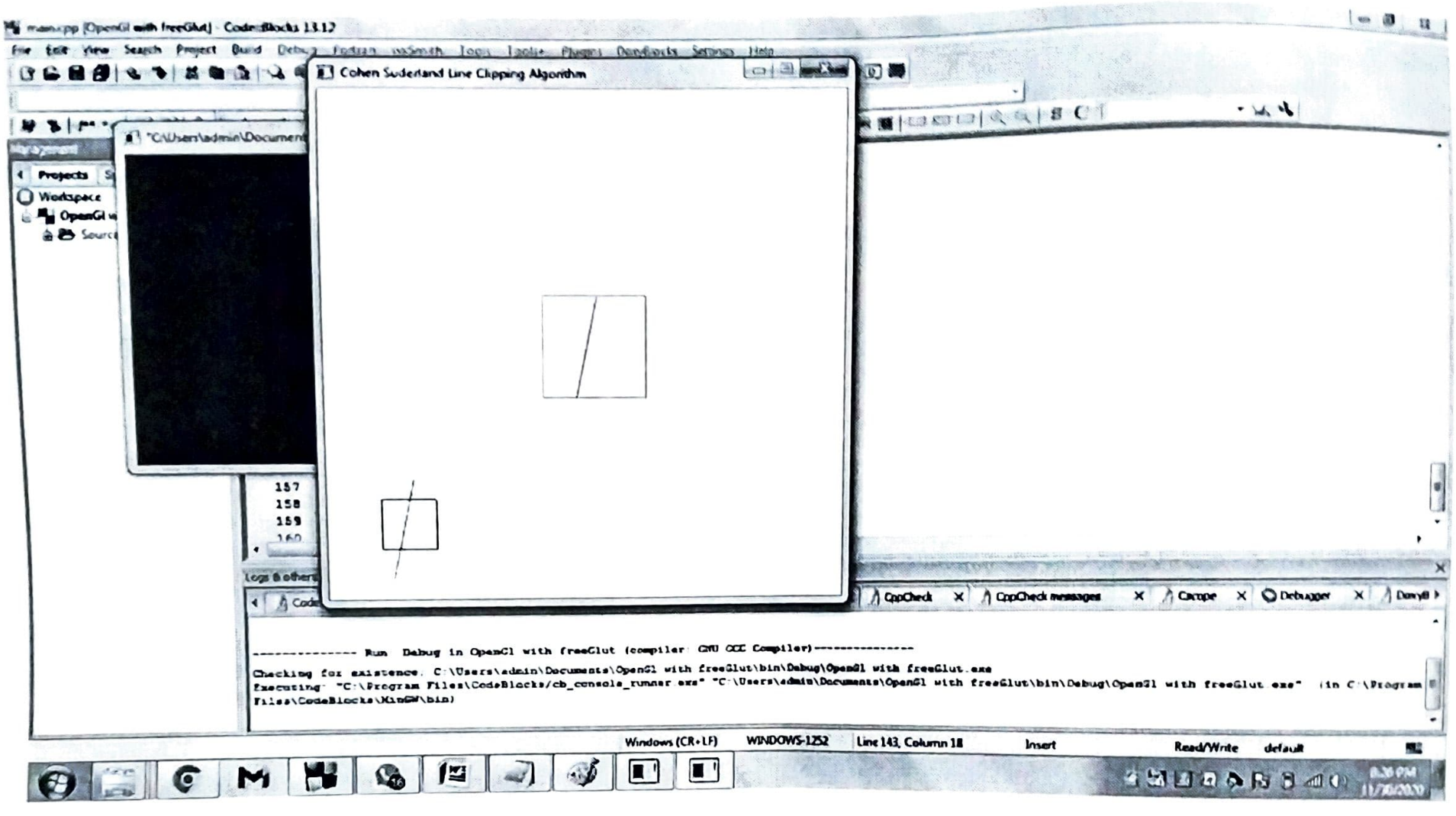
diagonally by (xmin, ymin) & (xmax, ymax)

```
//compute bitcode for a point (x,y) using clip rectangle bounded
outcode ComputeOutCode(double x, double y)
{ outcode code=0;
  if (y>ymax) // above the clip window
    code|=TOP;
  else if(y<ymin) // below the clip window
    code|=BOTTOM;
```

```
if(x>xmax) // to the right of the clip window
  code |= RIGHT;
else if (x<xmin) // to the left of clip window
  code |= LEFT;
  return code;
}
void display()
{  double x0=60, y0=20, x1=80, y1=120;
   glClear(GL_COLOR_BUFFER_BIT);
   // draw the line with red color
   glColor3f(1.0,0.0,0.0);
   glBegin(GL_LINES);
   glVertex2d(x0, y0);
   glVertex2d(x1,y1);
   glEnd();
   glColor3f(0.0,0.0,1.0);
   glBegin(GL_LINES-LOOP);
   glVertex2f(xmin, ymin);
   glVertex2f(xmax, ymin);
   glVertex2f(xmax,ymax);
   glVertex2f(xmin,ymax);
   glEnd();
   CohenSutherland(x0, y0, x1, y1);
   glFlush();
}  void myinit(){ . . . . . }
int main(int argc, char ** argv)
{  glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
   glutInitWindowSize(500,500);
```

```
glutInitWindowPosition(0,0);
glutCreateWindow("cohen sutherland line clipping algo");
glutDisplayFunc(display);
myInit();
glutMainLoop();
}
```

OUTPUT:


Cohen Sudertand Line Clipping Algorithm
Projects
Workspace
CppCheck
CppCheck messages
Cscope
Debugger
---------- Run: Debug in OpenGl with freeGlut (compiler: GNU GCC Compiler)----------
Checking for existence: C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe" (in C:\Program Files\CodeBlocks\MinGW\bin)
Windows (CR+LF)
WINDOWS-1252
Line 143, Column 18
Insert
Read/Write
default

PROGRAM 7

Write a program to implement the Liang-Barsky line clipping algorithm. Make provision to specify the i/p for multiple lines, window for clipping & viewport for displaying the clipped image.

```
#include<stdio.h>
#include<GL/glut.h>
double xmin=50, ymin=50, xmax=100, ymax=100; //window boundaries
double xvmin=200, yvmin=200, xvmax=300, yvmax=300; //viewport boundaries
int cliptest(double p, double q, double *t1, double *t2)
{ double t=q/p;
  if(p<0.0) // potentially entry point, update te
  { if(t>*t1) *t1=t;
    if(t>*t2) return(false) // line portion is outside
  }
  else
  if(p>0.0) // potentially leaving point, update tl
  { if(t<*t2) *t2=t;
    if(t<*t1) return(false); //line portion is outside
  }
  else
  if(p==0.0)
  { if(q<0.0) return(false); //line parallel to edge but outside
  }
  return(true);
}
```

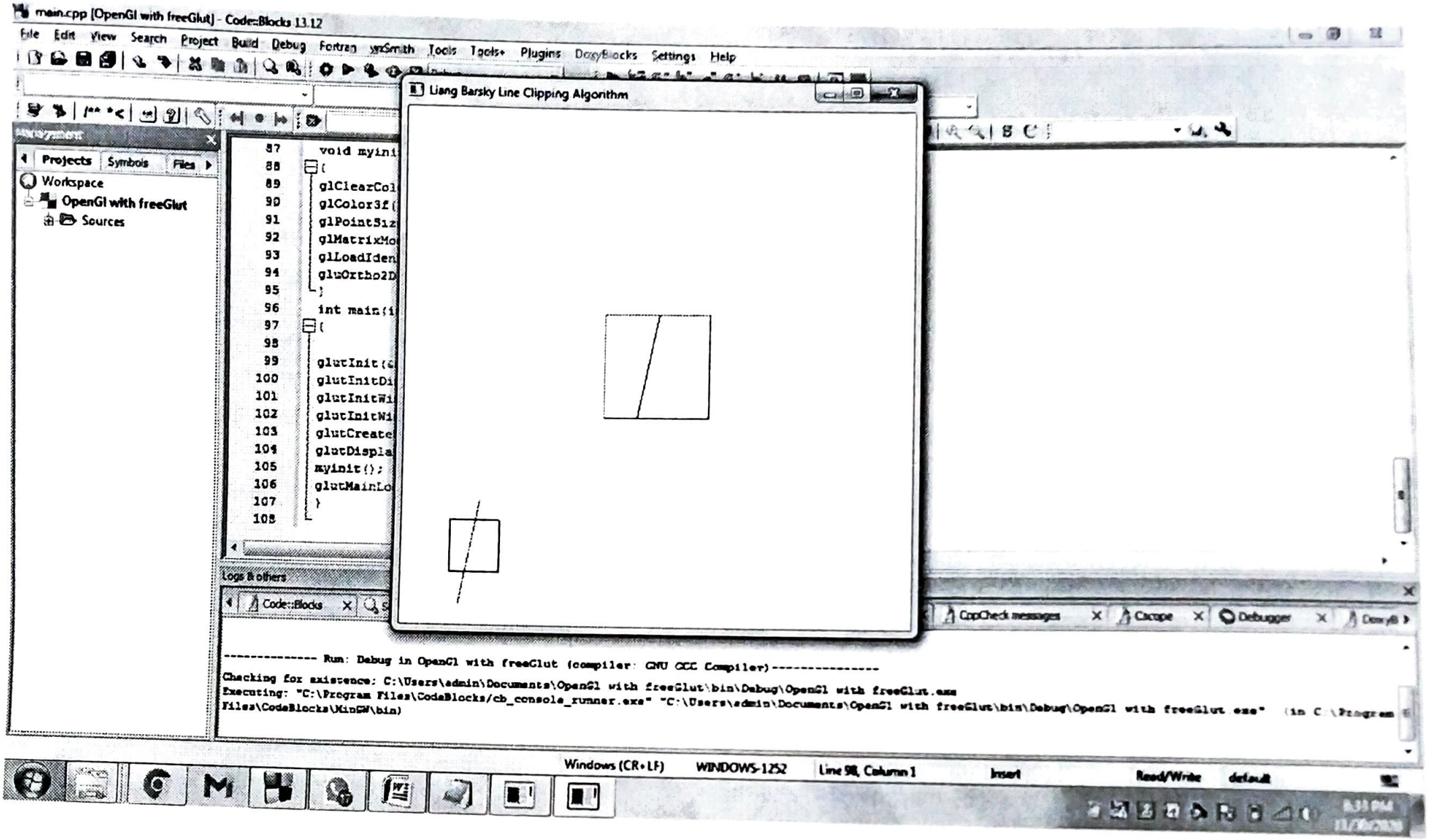
Name of Experiment ..... Date .....

Experiment No ..... Experiment Result ..... Page No. 27

```
void LiangBarskyLineClip(double x0, double y0, double x1, double y1)
{
    double dx = x1 - x0, dy = y1 - y0, te = 0.0, tl = 1.0;
    if(cliptest(-dx, x0 - xmin, &te, &tl)) // inside test wrt left edge
    if(cliptest(dx, xmax - x0, &te, &tl)) //  -  "  -  right edge
    if(cliptest(-dy, y0 - ymin, &te, &tl)) //  bottom
    if(cliptest(dy, ymax - y0, &te, &tl)) //  top
    {
        if(tl < 1.0)
        {
            x1 = x0 + tl * dx;
            y1 = y0 + tl * dy;
        }
        if(te > 0.0)
        {
            x0 = x0 + te * dx;
            y0 = y0 + te * dy;
        }

        double sx = (xvmax - xvmin) / (xmax - xmin);
        double sy = (yvmax - yvmin) / (ymax - ymin);
        double vx0 = xvmin + (x0 - xmin) * sx;
        double vy0 = yvmin + (y0 - ymin) * sy;
        double vx1 = xvmin + (x1 - xmin) * sx;
        double vy1 = yvmin + (y1 - ymin) * sy;
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(xvmin, yvmin);
        glVertex2f(xvmax, yvmin);
        glVertex2f(xvmax, yvmax);
        glVertex2f(xvmin, yvmax);
        glEnd();
        glColor3f(0.0, 0.0, 1.0);
        glBegin(GL_LINES);
        glEnd();
    }
}
```

chandra's

OUTPUT:


main.cpp [OpenGl with freeGlut] - Code::Blocks 13.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Liang Barsky Line Clipping Algorithm
Projects Symbols Files
Workspace
OpenGl with freeGlut
Sources
Logs & others
Code::Blocks
CppCheck messages
Cscope
Debugger
---------- Run: Debug in OpenGl with freeGlut (compiler: GNU GCC Compiler)----------
Checking for existence: C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe"
Files\CodeBlocks\MinGW\bin)
Windows (CR+LF)
WINDOWS-1252
Line 98, Column 1
Insert
Read/Write
default

Name of Experiment ..... Date .....

Experiment No. ..... Experiment Result ..... Page No. 28

PROGRAM 8

write a Program to implement Cohen-hodgeman polygon clipping algo. make provision to specify the input polygon & window for clipping.

```
#include<iostream>
#include<GL/glut.h>
#include<stdio.h>
using namespace std;
int poly_size, poly_points[20][20], org_size, org_poly_points[20][2],
clipped_size, clipped_points[20][2];
const int MAX_POINTS = 20;
void drawPoly(int p[][2], int n){
    glBegin(GL_POLYGON);
    for(int i=0; i<n; i++)
    glVertex2f(p[i][0],p[i][1]);
    glEnd();
}
int x_intersection(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;
}
int y_intersection(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;
```

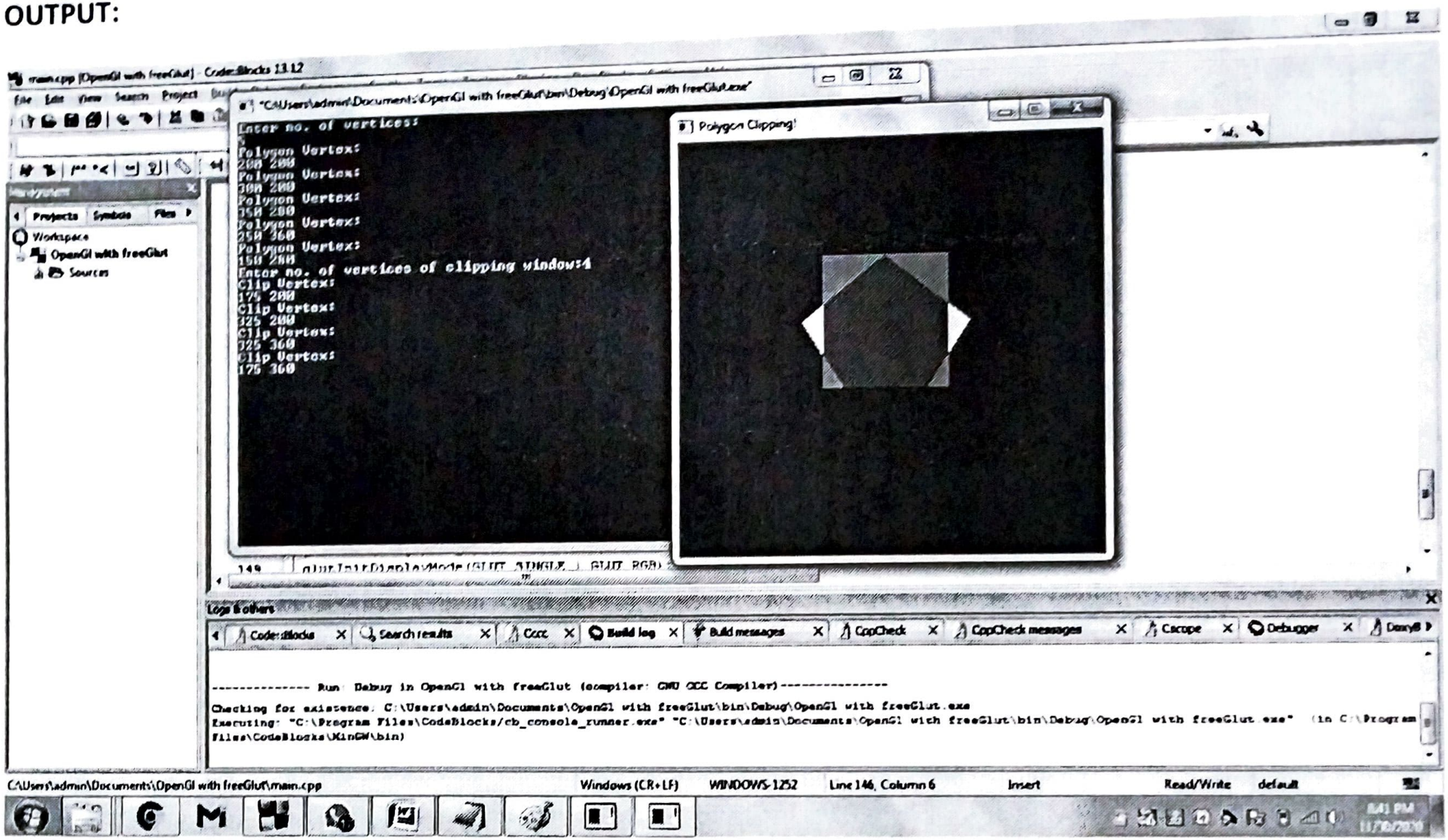
chandrat

```
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;
  for(int i=0; i<poly_size; i++){
    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];
    int i_pos = (x2-x1)*(iy-y1)-(y2-y1)*(ix-x1);
    int k_pos = (x2-x1)*(ky-y1)-(y2-y1)*(kx-x1);
    if(i_pos>=0 && k_pos>=0)
    {
      new_points[new_poly_size][0]=kx;
      new_points[new_poly_size][1]=ky;
    } new_poly_size++;
  else if(i_pos<0 && k_pos>=0)
  {
    new_points[new_poly_size][0] = x_intersect(x1,y1,x2,y2,ix,iy,kx,ky);
    new_points[new_poly_size][1] = y_intersection(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++; }
  else if(i_pos>=0 && k_pos<0){
    new_points[new_poly_size][0] = x_intersect(x1,y1,x2,y2,ix,iy);
    new_points[new_poly_size][1] = y_intersection(x1,y1,x2,y2,ix,iy);
    new_poly_size++; }
  else{ }}
  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]; }}
```

```
void display(){
    init();
    glColor3f(1.0f,0.0f,0.0f);
    drawPoly(clipper_points,clipper_size);
    glColor3f(0.0f,1.0f,0.0f);
    drawPoly(org_poly_points,org_poly_size);
    for(int i=0;i<clipper_size;i++)
    {   int k=(i+1)%clipper_size;
        clip(poly_points,poly_size,clipper_points[i][0],clipper_points[i][1],
        clipper_points[k][0],clipper_points[k][1]);}
        glColor3f(0.0f,0.0f,1.0f);
        drawPoly(poly_points,poly_size);
        glFlush();}

int main(int argc,char* argv[]){
    printf("Enter no of vertices");
    scanf("%d",&poly_size);
    org_poly_size=poly_size;
    for(int i=0;i<poly_size;i++){
        pf("Polygon vertex");
        sf("%d%d",&poly_points[i][0],&poly_points[i][1]);
        org_poly_points[i][0]=poly_points[i][0];
        org_poly_points[i][1]=poly_points[i][1];
    }
    pf("Enter no of vertices of clipping window");
    sf("%d",&clipper_size);
    for(int i=0;i<clipper_size;i++){
        pf("clip vertex");
        sf("%d%d",&clipper_points[i][0],&clipper_points[i][1]);}
```

```
glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400, 400);
glutInitWindowPosition(100,100);
glutCreateWindow ("polygon clipping");
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

OUTPUT:


Enter no. of vertices:
4
Polygon Vertex:
200 200
Polygon Vertex:
300 200
Polygon Vertex:
300 280
Polygon Vertex:
250 360
Polygon Vertex:
150 280
Enter no. of vertices of clipping window:4
Clip Vertex:
175 200
Clip Vertex:
325 200
Clip Vertex:
325 360
Clip Vertex:
175 360
Polygon Clipping
Workspace
OpenGl with freeGlut
Sources
Code::Blocks
Search results
Cccc
Build log
Build messages
CppCheck
CppCheck messages
Cscope
Debugger
Doxyg
Logs & others
---------- Run: Debug in OpenGl with freeGlut (compiler: GNU GCC Compiler)----------
Checking for existence: C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe
Executing: "C:\Program Files\CodeBlocks/cb_console_runner.exe" "C:\Users\admin\Documents\OpenGl with freeGlut\bin\Debug\OpenGl with freeGlut.exe" (in C:\Program Files\CodeBlocks\MinGW\bin)
C:\Users\admin\Documents\OpenGl with freeGlut\main.cpp
Windows (CR+LF)
WINDOWS-1252
Line 146, Column 6
Insert
Read/Write
default

PROGRAM - 09

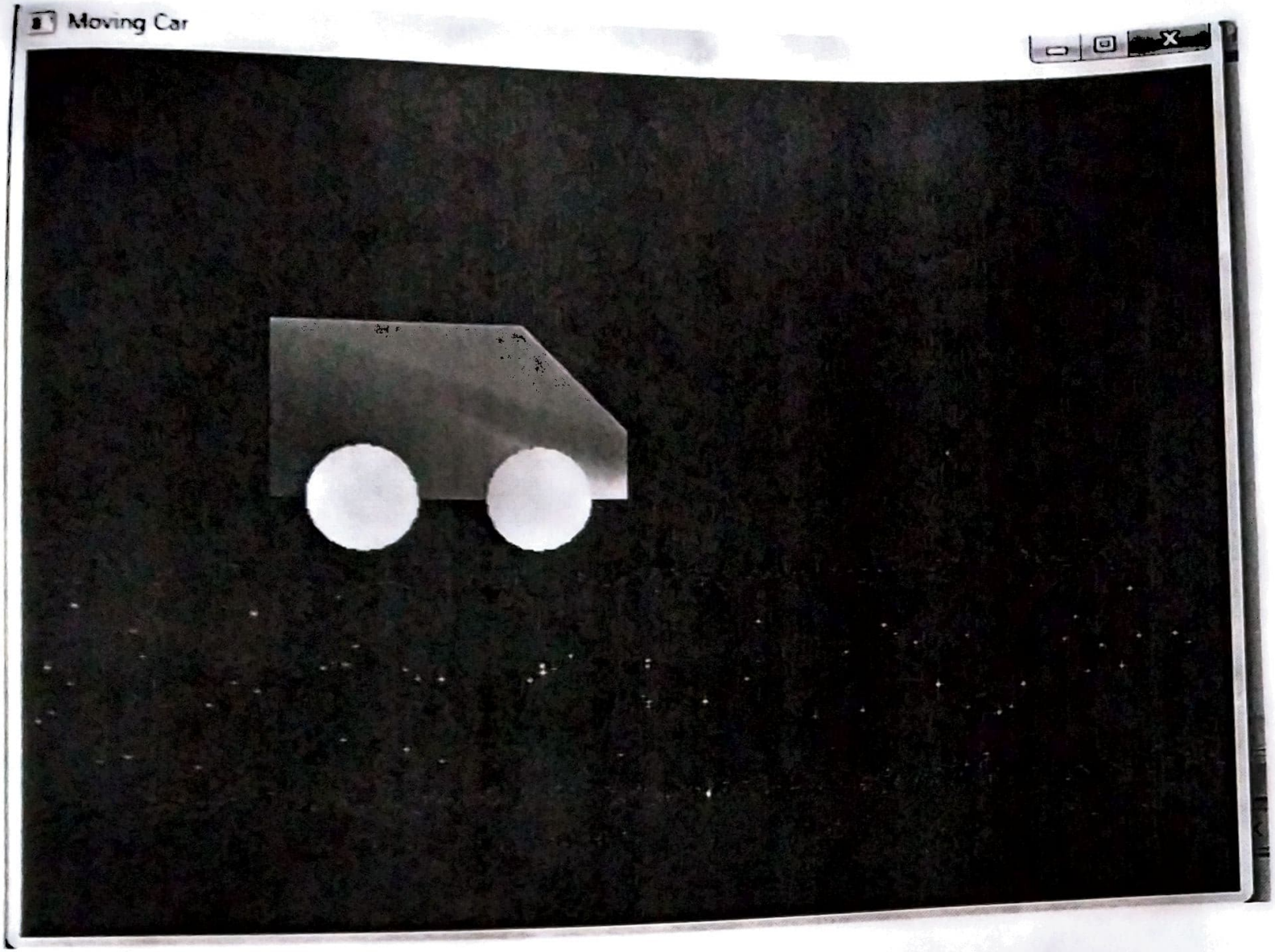
Write a Program to model car like figure using display lists and move a car from one end of the screen to other end. User is able to control the speed with mouse.

```
#include<GL/glut.h>
#include<math.h>
#include<stdio.h>
#define CAR 1
#define WHEEL 2
float s=1;
void carlist(){
      glNewList(CAR, GL_COMPILE);
      glColor3f(1,1,1);
      glBegin(GL_POLYGON);
      glVertex3f(0,25,0);
      glVertex3f(90,25,0);
      glVertex3f(90,55,0);
       glVertex3f(80,55,0);
       glVertex3f(20,75,0);
       glVertex3f(0,55,0);
     glEnd();
     glEndList(); }
void wheelList(){
      glNewList(WHEEL, GL_COMPILE_AND_EXECUTE);
      glColor3f(0,1,1);
      glutSolidSphere(10,25,25);
      glEndList();
```

```
void mykeyboard(unsigned char key, int x, int y){
        switch(key){
        case 'e': glutPostRedisplay();
                  break;
        case 'q': exit(0);
        default: break;
}
void myInit(){
        glClearColor(0,0,0,0);
        glOrtho(0,600,0,600,0,600);
}
void draw_wheel(){
        glColor3f(0,1,1);
        glutSolidSphere(10,25,25);
}
void moveCar(float s){
        glTranslatef(s,0.0,0.0);
        glCallList(CAR);
        glPushMatrix();
        glTranslatef(25,25,0.0);
        glCallList(WHEEL);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(75,25,0.0);
        glCallList(WHEEL);
        glPopMatrix();
        glFlush();
}
```

```
void myDispl(){
        glClear(GL_COLOR_BUFFER_BIT);
        carlist();
        movecar(s);
        wheellist();
}

void mouse(int btn, int state, int x, int y){
        if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN){
                st = 5;
                myDispl();
        }else if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN){
                st = 2;
                myDispl();
        }
}

int main(int argc, char *argv[]){
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(600,500);
        glutInitWindowPosition(100,100);
        glutCreateWindow("car");
        myInit();
        glutDisplayFunc(myDisp);
        glutMouseFunc(mouse);
        glutKeyboardFunc(myKeyboard);
        glutMainLoop();
}
```

OUTPUT:



PROGRAM - 10

Write a program to create a color cube and spin it using OpenGL Transformations.

```
#include<stdlib.h>
#include<GL/glut.h>
#include<gl/GL.h>
#include<gl\GLU.h>
#include<time.h>
GLfloat vertices[] = {-1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0,-1.0,1.0,-1.0
-10, 1.0, 1.0,-1.0, 1.0,1.0, 1.0, 1.0, -1.0, 1.0, 1.0};
GLfloat normals[] = {-1.0,-1.0,1.0,1.0,-1.0,-1.0, 1.0,1.0,-1.0,-1.0,-1.0,
1.0, 1.0,-1.0, 1.0,1.0, 1.0,-1.0,1.0,1.0, 1.0,-1.0, 1.0,1.0};
GLfloat colors[] = {0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,0.1,1.0,0.0,1.0
GLubyte cubeIndices[] = {0,3,2,1,2,3,7,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4};
static GLfloat theta[] = {0.0,0.0,0.0};
static GLfloat beta[] = {0.0,0.0,0.0};
static GLint axis = 2;
void delay(float secs)
{   float end = clock() / CLOCKS_PER_SEC + secs;
    while((clock() / CLOCKS_PER_SEC) < end);
}
void displaySingle(void)
{   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);
```

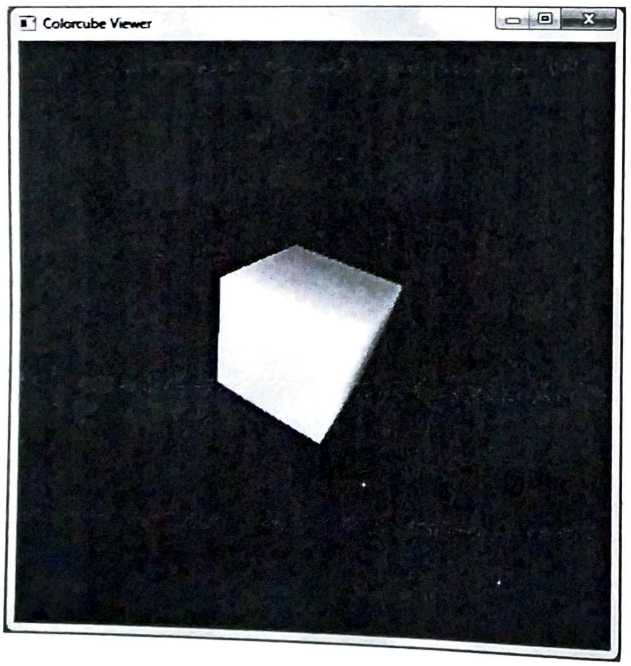
```
    glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, useindices);
    glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.0, 1.0, 1.0);
    glEnd();
    glFlush();
}
void spincube()
{
    delay(0.01);
    theta[axis] += 2.0;
    if(theta[axis] > 360.0) theta[axis] -= 360.0;
    glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y)
{
    if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
    if(btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
    if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}
void myReshape(int w, int h)
{
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w <= h)
        glOrtho(-2.0, 2.0, -2.0 * (GLfloat)h/(GLfloat)w,
                2.0 * (GLfloat)h/(GLfloat)w, -10.0, 10.0);
    else
        glOrtho(-2.0 * (GLfloat)w/(GLfloat)h,
                2.0 * (GLfloat)w/(GLfloat)h, -2.0, 2.0, -10.0, 10.0);
    glMatrixMode(GL_MODELVIEW);
```

```
void main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(500,500);
    glutCreateWindow("colorcube");
    glutReshapeFunc(myReshape);
    glutDisplayFunc(displaySingle);
    glutIdleFunc(spinCube);
    glutMouseFunc(spinCube);
    glutMouseFunc(mouse);
    glEnable(GL_DEPTH_TEST);
    glEnableClientState(GL_COLOR_ARRAY);
    glEnableClientState(GL_NORMAL_ARRAY);
    glVertexPointer(3, GL_FLOAT, 0, vertices);
    glColorPointer(3, GL_FLOAT, 0, colors);
    glNormalPointer(GL_FLOAT, 0, normals);
    glColor3f(1.0, 1.0, 1.0);
    glutMainLoop();
}
```

Output:

Rotation through X axis

Perspective view with Key 'x'



Perspective view with Key 'y'

PROGRAM - 11

Create a menu with three entries named curves, colors & quit. The entry curves has a sub menu which has four entries namely Limacon, Cardiod, Three-Leaf, and spiral. The color menu has sub menu with all eight colors of RGB color model. Write program to create the above hierarchical menu and attach appropriate services to each menu entries with mouse buttons.

```
#include<gl/glut.h>
#include<math.h>
#include<stdio.h>
struct screenPt{
    int x; int y;
};
typedef enum{limacon=1, cardiod=2, threeleaf=3, spiral=4}curvename;
int w=600, h=500;
int curve=1;
int red=0, green=0, blue=0;
void myinit(void){
    glClearColor(1.0,1.0,1.0,1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}
void lineSegment(screenPt P1, screenPt P2){
    glBegin(GL_LINES);
    glVertex2i(P1.x, P1.y);
    glVertex2i(P2.x, P2.y);
    glEnd();
    glFlush();
}
```

```
void drawCurve(int curveNum){
    const double twoPi = 6.283185;
    const int a=175, b=60;
    float r, theta, dtheta=1.0/float(a);
    int x0=200, y0=250;
    screenPt curvePt[2];
    curve= curveNum;
    glColor3f (red, green, blue);
    curvePt[0].x= x0;
    curvePt[0].y = y0;
    glClear (GL_COLOR_BUFFER_BIT);
    switch (curveNum){
      case limacon: curvePt[0].x+= a+b; break;
      case cardioid: curvePt[0].x+= a+a; break;
      case threeLeaf: curvePt[0].x+= a; break;
      case spiral: break;
      default: break;
    }
    theta = dtheta;
    while (theta < twoPi){
      switch (curveNum){
      case limacon: r= a* cos(theta)+ b; break;
      case cardioid: r= a*(1+cos(theta)); break;
      case threeLeaf: r= a* cos(3* theta); break;
      case spiral: r= (a/4.0)* theta; break;
      default: break;
    }
    curvePt[1].x= x0+ r* cos(theta);
    curvePt[1].y= y0+ r* sin(theta);
```

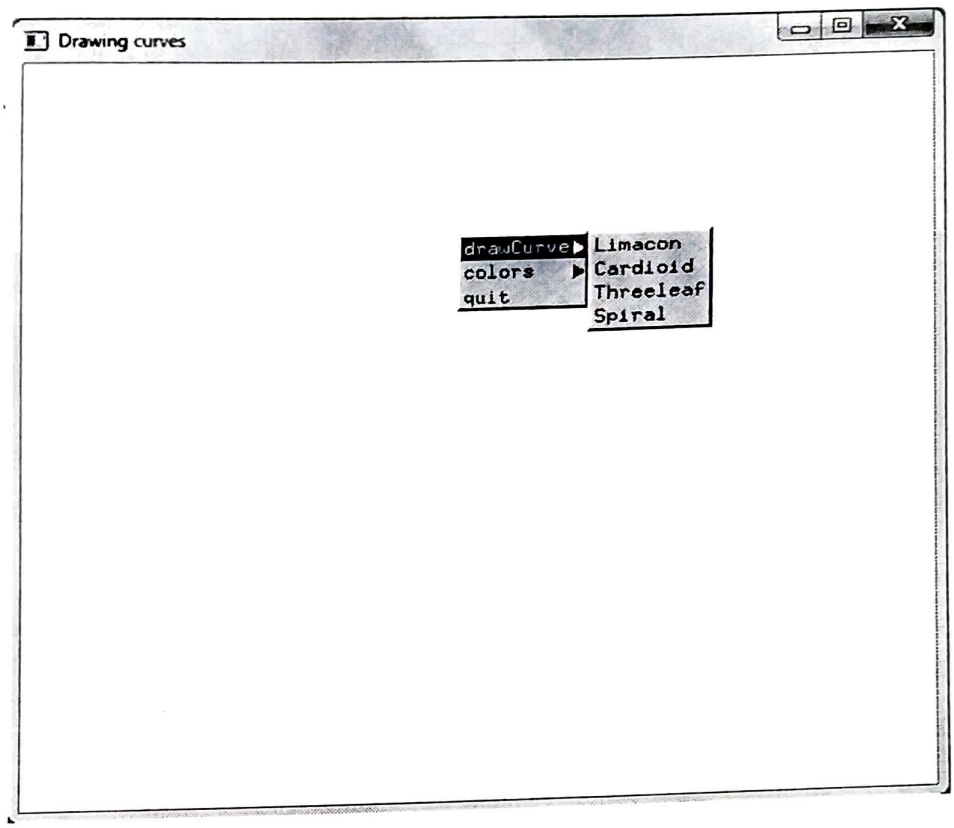
```
        linesegment(curvePt[0], curvePt[1]);
        curvePt[0].x = curvePt[1].x;
        curvePt[0].y = curvePt[1].y;
        theta = dtheta;
    }
}
void colormenu(int id){
    switch(id){
    case 0:
        break;
    case 1:
        red=0;
        green=0;
        blue=1;
        break;
    case 2:
        red=0;
        green=1;
        blue=0;
        break;
    case 3: red=0;
        green=1
        blue=1;
        break;
    case4: red=1;
        green=0;
        blue=0;
        break;
    case 5: red=1;
        green=0
        blue=1
        break;
    case 6 : red=1;
        green=1;
        blue=0;
        break;
    case 7: red=1;
        green=1;
        blue=1;
        break;
    default: break;
    }
    drawcurve(curve);
}
void main_menu(int id){
    switch(id){
    case 3: exit(0);
    default: break;
    }
}
```

```
void mydisplay(){
    glClear(GL_COLOR_BUFFER_BIT);
}
void myreshape(int nw, int nh){
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, (double)nw, 0.0, (double)nh);
    glClear(GL_COLOR_BUFFER_BIT);
}
void main(int argc, char** argv){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(w,h);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Drawing Curves");
    int curveid = glutCreateMenu(drawcurve);
    glutAddMenuEntry("Limacon", 1);
                    ("cosdioid", 2);
                    ("Threeleaf", 3);
                    ("spiral", 4);
    glutAttachMenu(GLUT_LEFT_BUTTON);
    int colorid = glutCreateMenu(colormenu);
    glutAddMenuEntry("Red", 4);
    glutAddMenuEntry("Green", 2);
                    ("Blue", 1);
                    ("Black", 0);
                    ("yellow", 6);
                    ("cyon", 3);
                    ("magenta", 5);
```

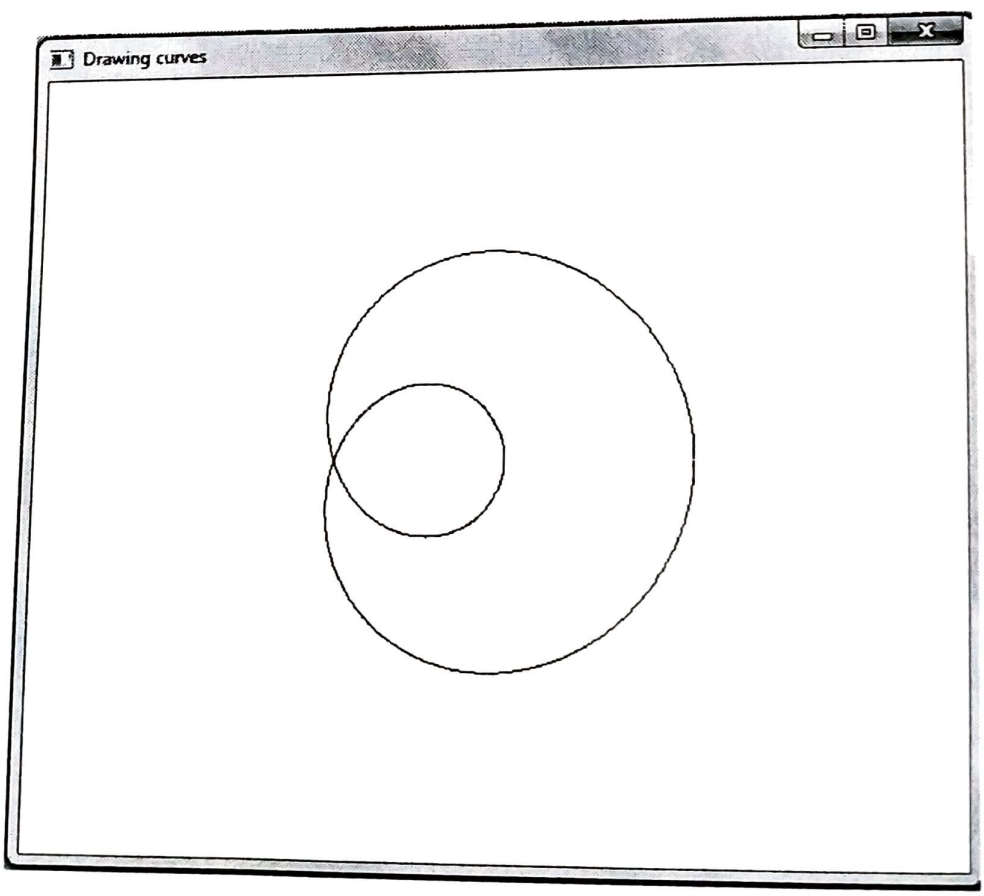
```
glutAddMenuEntry ("White", 7);
glutAttachMenu (GLUT_LEFT_BUTTON);
glutCreateMenu (main_menu);
glutAddSubMenu ("drawCurve", curveid);
glutAddSubMenu ("colors", colorid);
glutAddMenuEntry ("quit", 3);
glutAttachMenu (GLUT_LEFT_BUTTON);
myinit ();
glutDisplayFunc (mydisplay);
glutReshapeFunc (myreshape);
glutMainLoop ();
}
```

Output:

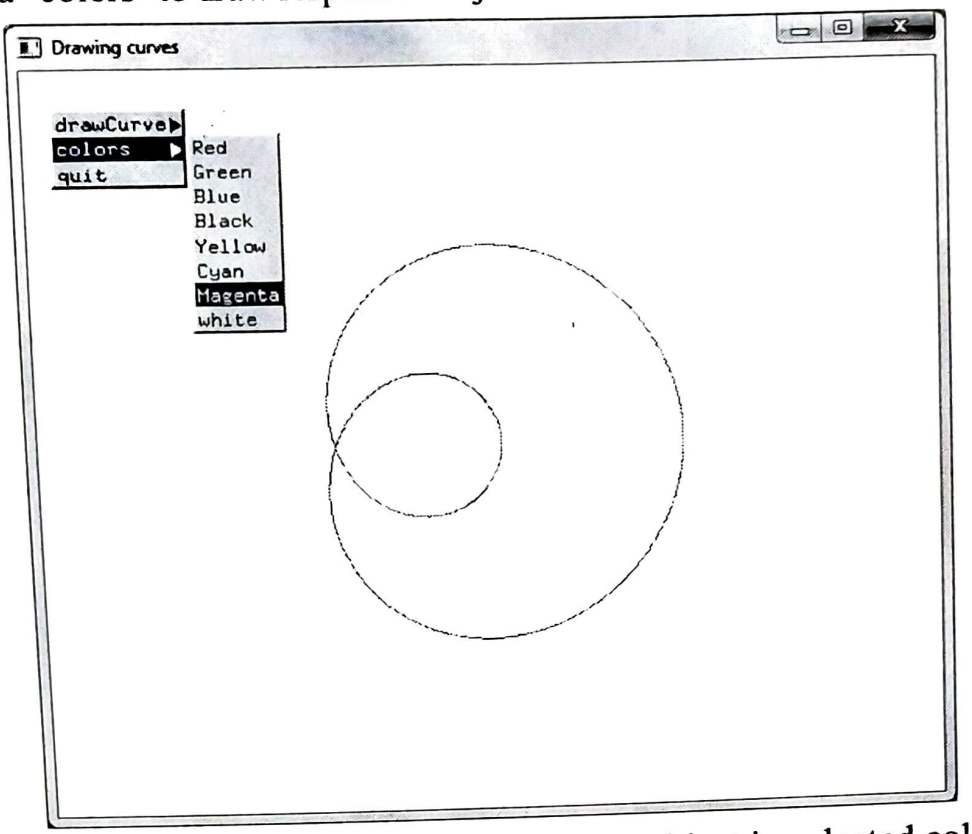
- Click left button show and select the menu.



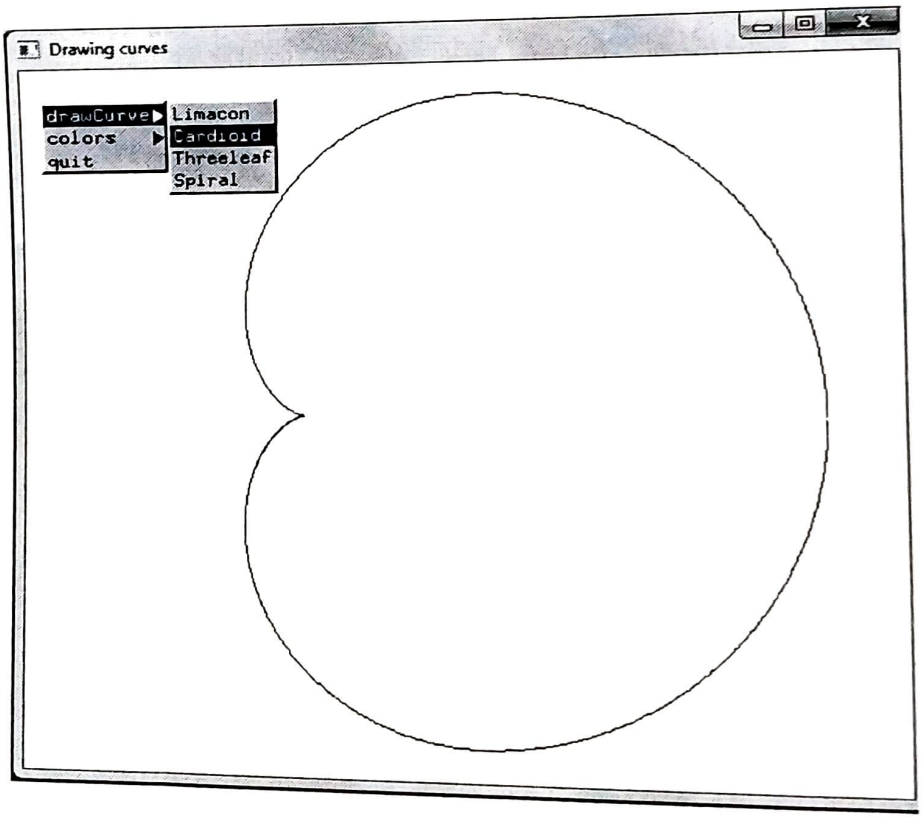
- Choose menu options to draw respective objects.

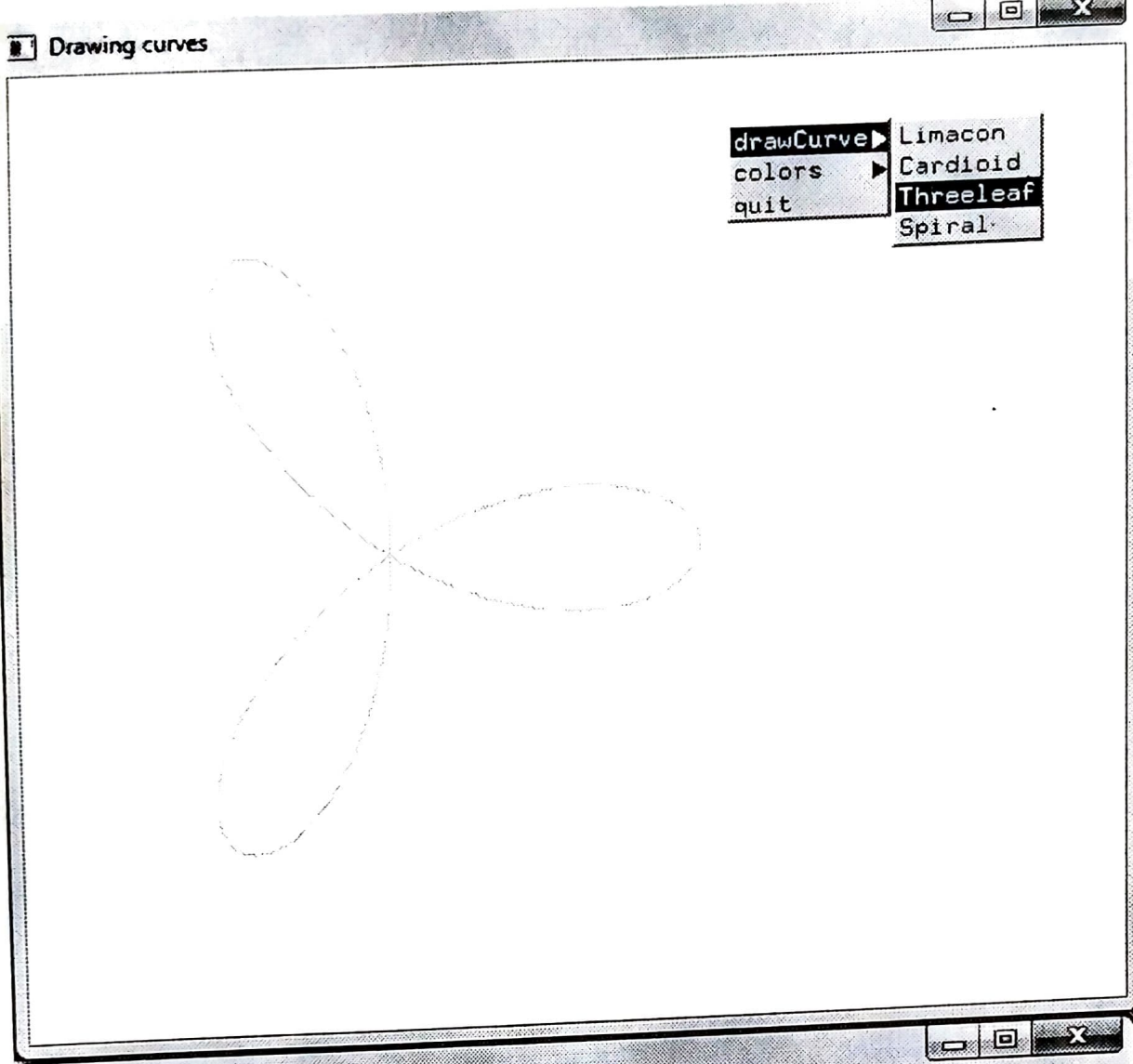


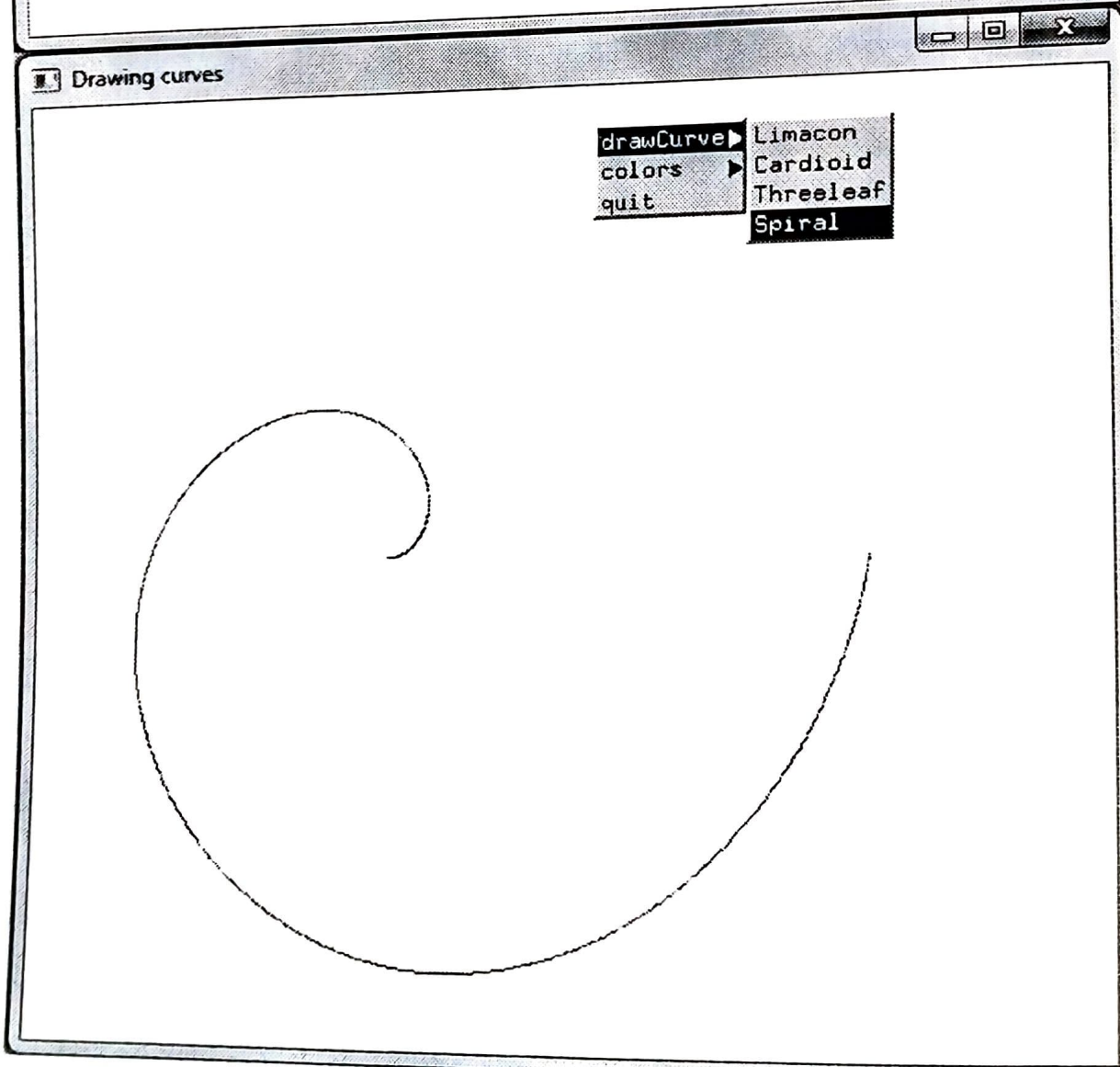
- Choose submenu 'colors' to draw respective objects in the selected color(within eight colors).



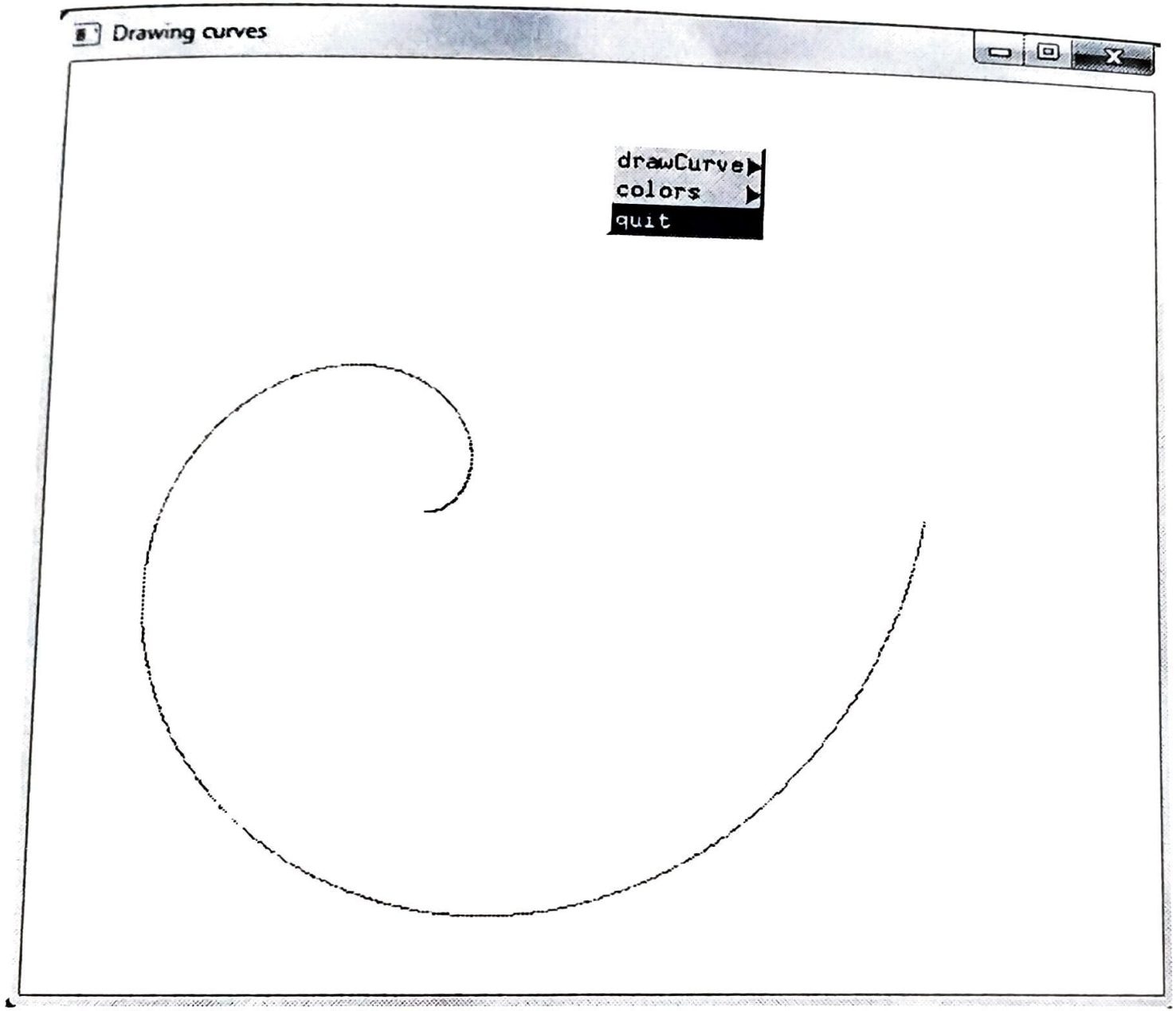
- Select limacon/cardioid/threeleaf/spiral to draw Respective object in selected color.




Drawing curves
drawCurve
colors
quit
Limacon
Cardioid
Threeleaf
Spiral


Drawing curves
drawCurve
colors
quit
Limacon
Cardioid
Threeleaf
Spiral

select quit to exit.



PROGRAM-12

Write a program to construct Beziers curve. Control points are supplied through keyboard/mouse.

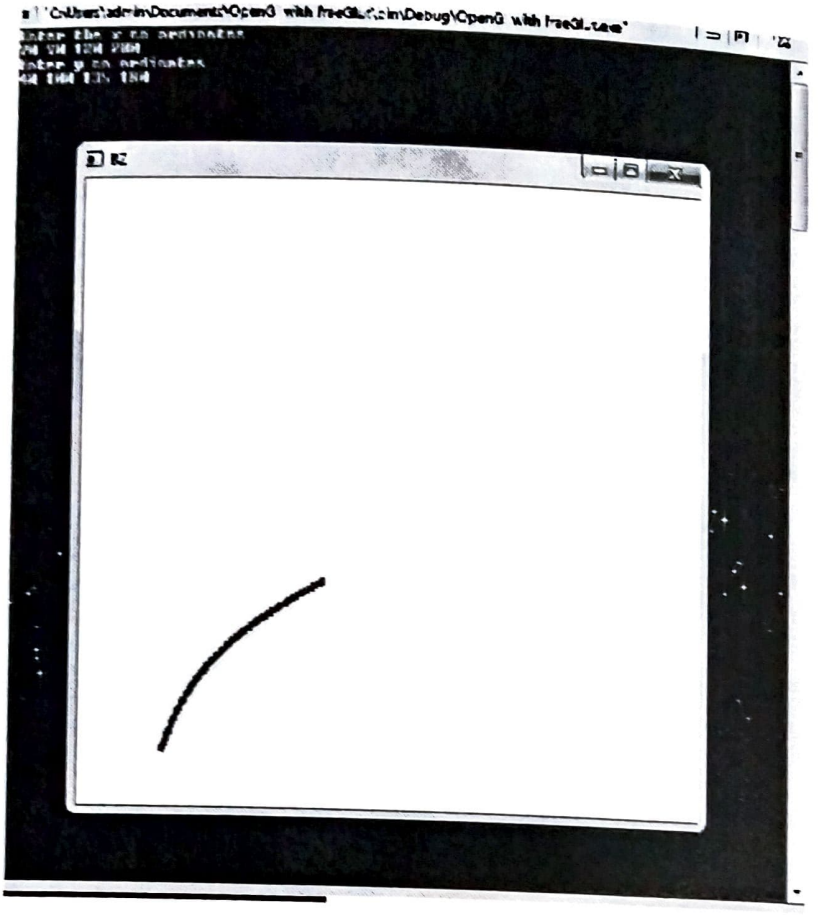
```
#include<iostream>
#include<math.h>
#include<gl/glut.h>
using namespace std;
float f, g, r, x1[4], yc[4];
int flag=0;
void myInit(){
        glClearColor(1,1,1,1);
        glColor3f(1,1,1);
        glPointSize(5);
        gluOrtho2D(0,500,0,500);
}
void drawPixel(float x, float y){
        glBegin(GL_POINTS);
        glVertex2f(x,y);
        glEnd();
}
void display(){
        glClear(GL_COLOR_BUFFER_BIT);
        int i;
        double t;
        glColor3f(0,0,0);
        glBegin(GL_POINTS);
        for(t=0; t<1; t=t+0.005){
```

```
    double xt = pow(1-t,3) * x1[0] + 3 * t * pow(1-t,2) * x1[1] + 3 *
                pow(t,2) * (1-t) * x1[2] + pow(t,3) * x1[3];
    double yt = pow(1-t,3) * yc[0] + 3 * t * pow(1-t,2) * yc[1] + 3 *
                pow(t,2) * (1-t) * yc[2] + pow(t,3) * yc[3];
    glVertex2f(xt, yt);
  }
  glColor3f(1,1,0);
  for(i=0; i<4; i++){
      glVertex2f(x1[i], yc[i]);
      glEnd();
      glFlush();
    }
}
void mymouse(int btn, int state, int x, int y)
{ if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN && flag<4)
    {   x1[flag] = x;
        yc[flag] = 500 - y;
        cout << "x" << x << "y" << 500 - y;
        glPointSize(3);
        glColor3f(1, 0, 0);
        glBegin(GL_POINTS);
        glVertex2i(x, 500, -y);
        glEnd();
        glFlush();
        flag++;
    }
    if (flag >= 4 && btn == GLUT_LEFT_BUTTON)
    {       glColor3f(0,0,1);
            display();
            flag=0;
    }
}
```

```
int main(int argc, char* argv[]){
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("BZ");
    glutDisplayFunc(display);
    glutMouseFunc(myMouse); //include for mouse
    //use keyboard
    cout<<"enter the x coordinate";
    cin>>x1[0]>>x1[1]>> x1[2]>>x1[3];
    cout<<"enter y coordinate";
    cin>>y1[0]>>y1[1]>>y1[2]>>y1[3];
    //end keyboard
    myInit();
    glutMainLoop();
}
```

Output:

- Drawing BZ curve by inputting control points through keyboard.



- Drawing BZ curve through Mouse