Certificate

Name: S. Suraj Class: VII C Batch C2

Roll No: 1RV17CS130 Exam No:

Institution RV. COLLEGE OF ENGINEERING

This is certified to be the bonafide work of the student in the Computer Graphics *Laboratory during the academic year 2020 / 2021 .*

No. of practicals certified _____ *out of* _____ *in the subject of* _____

.....
Teacher In-charge

.....
Examiner's Signature

.....
Principal

Date: .....

Institution Rubber Stamp

(N.B: The candidate is expected to retain his/her journal till he/she passes in the subject.)

Index

Index

S. No.	Name of the Experiment	Page No.	Date of Experiment	Date of Submission	Remarks
11.	Program to create the specified hierarchical menu and attaching	50-56			
12.	Program to construct Bezier Curve	57-60			

Date _____

Expt. No. _____ Page No. 1

PROGRAM - 1

Q 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>
  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;
```

Teacher's Signature _____

```
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;
  incl = 2*(dy - dx);
  incl = 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;
    incl = 2 * (dx - dy);
    inc2 = 2 * dx;
    for (i=0; i<dy; i++
    {
        if (e>0)
        {
            x += incx;
            e += incl;
        }
        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;
                y1 = 250 - y;
                flag++;
                cout << x1 << " " << y1 << \n";
            }
            else
            {
                printf ("Defining x2, y2");
                x2 = x - 250;
                y2 = 250 - y;
                flag = 0;
                cout << x2 << " " << y2 << "\n";
                draw_line();
            }
        }
        break;
    }
}
void display()
{}
```

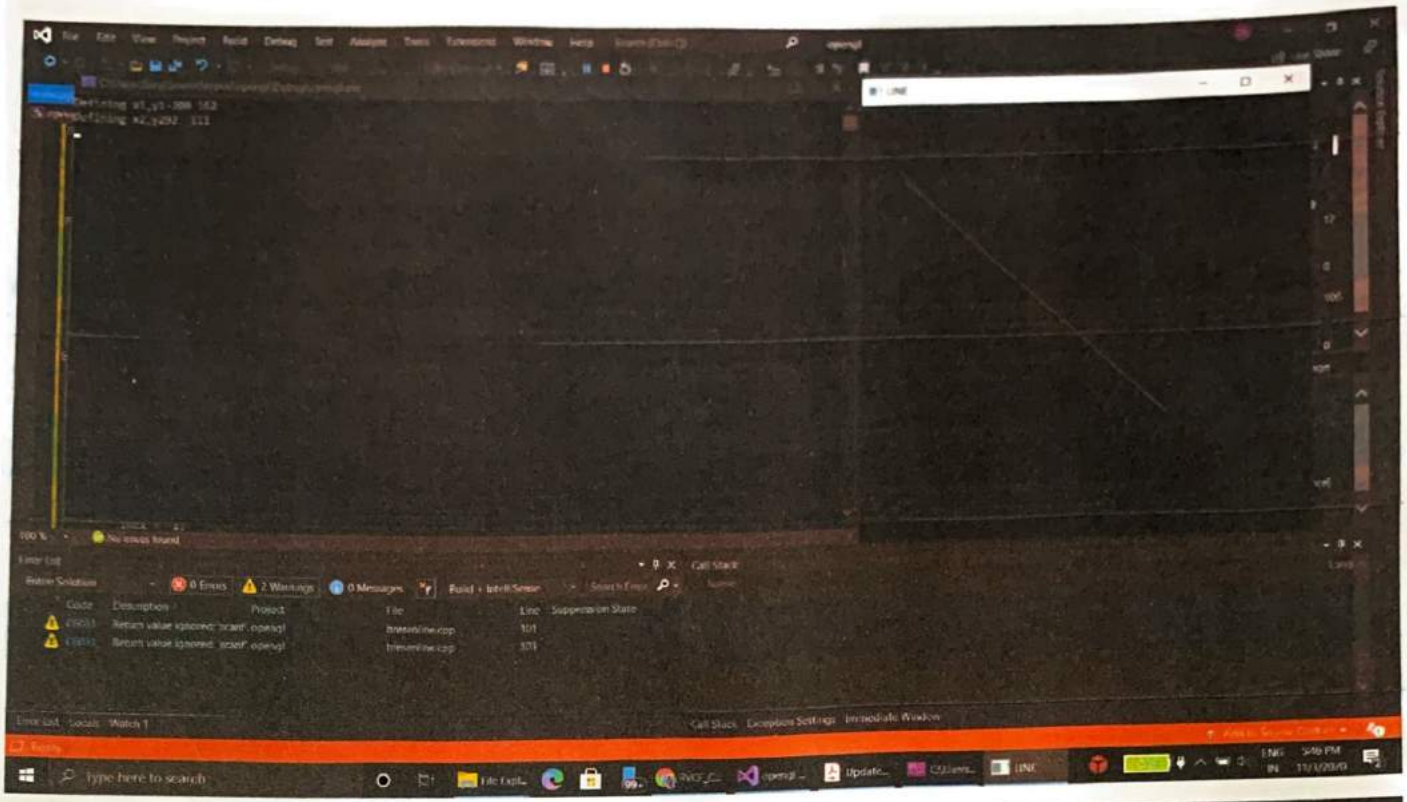
```
int main(int ac, char* av[])
{
    /*
    //FOR KEYBOARD
    cout<<"x1\n";
    cin>>x1;
    cout<<"y1\n";
    cin>>yc1;
    cout<<"x2\n";
    cin>>x2;
    cout<<"y2\n";
    cin>>y2;
    //END KEYBOARD
    */
    glutInit(&ac,av);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,200);
    glutCreatWindow("LINE")
    Myinit();
    glutMouseFunc(Mymouse); //INCLUDE to use MOUSE, REMOVE
                            WHILE USING MOUSE
    glutDisplayFunc(display);
    glutMainLoop();
```

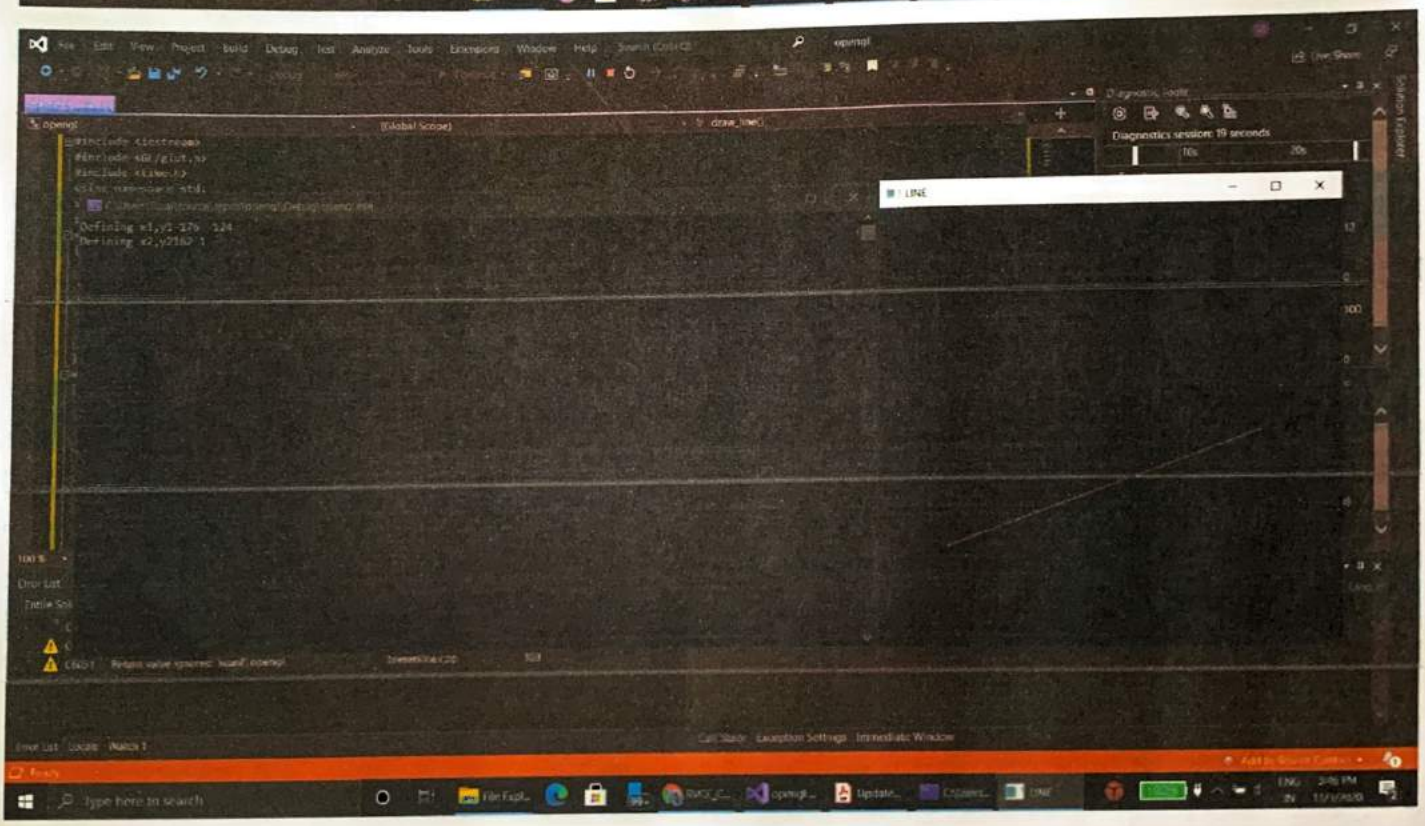
program 1

```
#include<iostream>
#include<GL/glut.h>
#include<time.h>
usingnamespace std;
int x1, x2, yc1, y2;
int flag = 0;

voiddraw_line()
{
        int dx, dy, i, e;
        intincx, 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();
}
voiddraw_pixel(intx, inty)
{
        glColor3f(1, 0, 0);
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        glEnd();
        glFlush();

}

voidmyinit()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1, 1, 1, 1);
```

```
        gluOrtho2D(-250, 250, -250, 250);
}

voidMyMouse(intbutton, intstate, intx, inty)
{
        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;
        }
}
voiddisplay()
{}
intmain(intac, char* av[])
{

        //KEYBOARD
        cout<<"X1\n";
        cin>>x1;
        cout<<"Y1\n";
        cin>>yc1;
        cout<<"X2\n";
        cin>>x2;
        cout<<"Y2\n";
        cin>>y2;

        glutInit(&ac, av);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 200);
        glutCreateWindow("LINE");
        myinit();
        glutMouseFunc(MyMouse);
        draw_line();
        glutDisplayFunc(display);
        glutMainLoop();
}
```

output





PROGRAM-2

Q 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, xce, yce;
Void draw - circle (int xc, int yc, int x, int y)
{
    glBegin (GL - POINTS);
    gl Vertex2i (xc+x,yc + y);
    gl vertex 2i (xc-x,yc + y);
    gl Vertex 2i (xc + x, yc-y);
    gl Vertex 2i (xc - x, yc-y);
    gl Vertex 2i (xc+y, yc+x);
    gl Vertex 2i (xc-y; yc+x);
    gl Vertex 2i (xc+y; yc-x);
    gl Vertex 2i (xc-y, yc-x);
    gl End ();
}
void circlebres()
{
    gl clear (GL - COLOR - BUFFER - BIT);
```

Teacher's Signature _____

```
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 ////////////
void draw-ellipse(int xce, int yce, int x, int y)
{
    gl Begin (GL-POINTS);
    gl Vertex2i (x+xce, y + yce);
    gl Vertex 2i (-x +xce, -y + yce);
    gl Vertex 2i (x+xce, -y +yce);
    gl vertex 2i (-x + xce, -y+ yce);
    gl End ();
}
void midptellipse()
{
    gl clear (GL-COLOR-BUFFER-BIT);
    float dx, dy, d1, d2, x, y;
    X = 0
    y = ry;
```

Teacher's Signature _____

```
// Initial decision Parameter of region 1
d1 = (ry * ry) - (rx * rx * ry) +
        (0.25 * rx * rx);
dx = 2 * ry * ry * x;
dy = 2 * rx * rx * y;

// For region 1
while (dx < dy)
{
    // Point points based on 4-way Symmetry
    draw - Ellispse(xce, yce, x, y);

    // checking and updating value of
    // decision parameter based on algorithm
    if (d1 < 0)
    {
        x++;
        dx = dx + (2 * ry * ry);
        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
drawEllipse(xce, yce, x, y);

// Checking and uploading Parameter
// value based on algorithm
if (d2 > 0)
{
    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, p2e_y, p3e_x, p3e_y;
int pointle = done = 0;
void myMouseFunc(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN
&& pointle_done == 0)
    {
        p1e_x = x - 250;
        p1e_y = 250 - y;
        xce = p1e_x;
        yce = p1e_y;
        pointle_done = 1;
    }
```

```
else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && pointle_done == 1)
{
        p2e_x = x - 250;
        p2e_y = 250 - y;
        float exp = (p2e_x - ple_x)*(p2e_x - ple_x) + (p2e_y - ple_y)*(p2e_y - ple_y);
        rx = (int)(sqrt(exp));
        // midptellipse();
        pointle_down = 2;
}
else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && pointe_done == 2)
{
        p3e_x = x - 250;
        p3e_y = 250 - y;
        float exp = (p3e_x - ple_x)*(p3e_x - ple_x) + (p3e_y - ple_y)*(p3e_y - ple_y);
        ry = (int)(sqrt(exp));
        midptellipse();
        pointle_done = 0;
}
}
void myDrawing()
{ }
Void myDrawing()
{ }
Void minit()
```

```
{
    glClearColor(1.1,1.1,1.1);
    glColor3f(1.0, 0.0, 0.0);
    glPointSize(3.0);
    gluOrtho2D(-250, 250, -250, 250);
}
void main(int argc, char *argv[])
{
    glutInit(&argc, argv);
    glutInitDisplaymode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0,0);
    //*
    //FOR MOUSE
    int id1 = glutCreatWindow("circle");
    glutSetWindow(id1);
    glutMouseFunc(myMouseFuncCircle);
    glutDisplayFunc(myDrawing);
    minit();
    glutInitWindowSize(500,500);
    glutInitWindowPosition(600,100);
    int id2 = glutCreatWindow("Ellipse");
    glutSetWindow(id2);
    glutMouseFunc(MyMouseFunc);
    glutDisplayFunc(MyDrawing);
    // END MOUSE
    */
    // FOR KEYBOARD
    printf("Enter 1 to draw circle, 2 to draw ellipse\n");
```

Teacher's Signature _____

```
int ch;
Scanf("%d",&ch);
Switch(ch){
case 1;
Print f(*Enter coordinates of centre of circle and radius\n*");
Scan f("%d%d%d",&xc,&yc,&r);
glut creat window("circle");
glut Display Func (circlebres);
 Case 2;
Print f("Enter coordinates of centre of ellipse and major
and minor radius\n*");
 Scan f("%d%d%d%d",&xce,&yce,&rx,&ry);
 glut creat window("Ellipse");
 glut display Func (midptellipse);
 break;
 }
 //END KEYBOARD
 minit();
 glut MainLoop();
 }
```

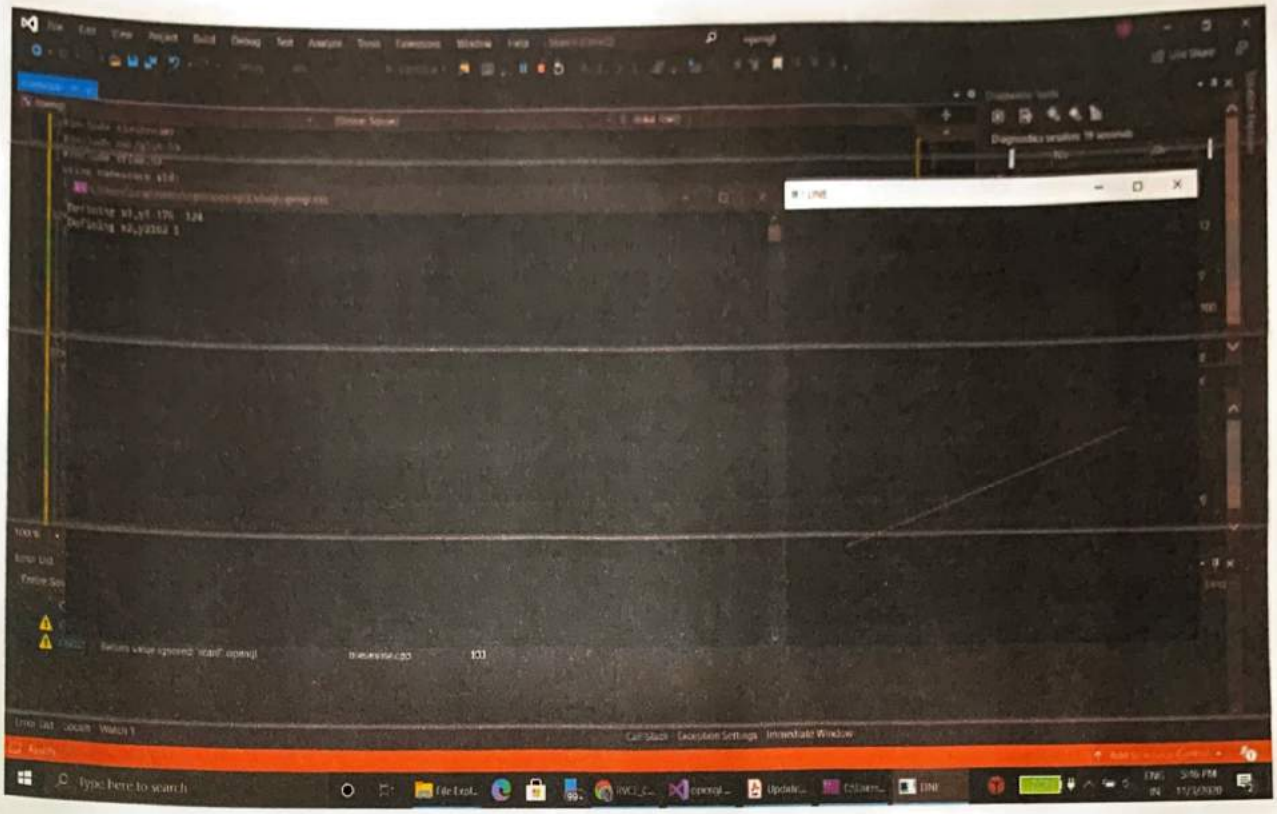
program 2

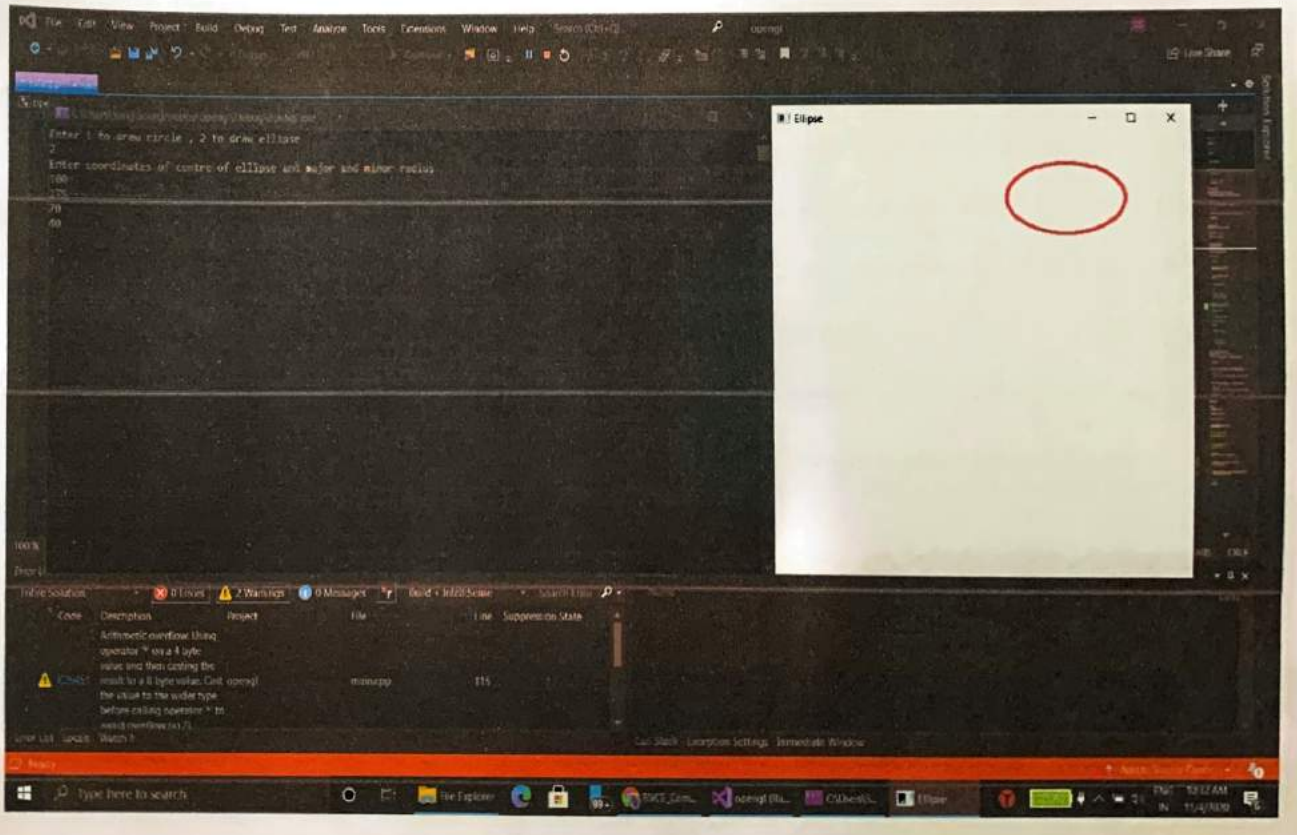
```
#include<gl/glut.h>
#include<stdio.h>
#include<math.h>
int xc, yc, r;
intrx, ry, xce, yce;

//Circle
voiddraw_circle(intxc, intyc, intx, inty)
{
        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();
}

//Generate circle using Bresenham's circle drawing algorithm
voidcirclebres()
{
        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;
voidmyMouseFunccircle(intbutton, intstate, intx, inty)
{
        if (button == GLUT_LEFT_BUTTON &&state == GLUT_DOWN && point1_done == 0)
        {
                p1_x = x - 250;
                p1_y = 250 - y;
                point1_done = 1;
        }
        elseif (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
voiddraw_ellipse(intxce, intyce, intx, inty)
{
        glBegin(GL_POINTS);
        glVertex2i(x + xce, y + yce);
        glVertex2i(-x + xce, y + yce);
        glVertex2i(x + xce, -y + yce);
        glVertex2i(-x + xce, -y + yce);
        glEnd();
}

//Generate ellipse using Bresenham's ellipse drawing algorithm
voidellipsebres()
{
        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);
                // 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 (slope>1)
        while (y >= 0)
        {
                // Print points based on 4-way symmetry
                draw_ellipse(xce, yce, x, y);

                // 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);
                }
        }
        glFlush();
}


int p1e_x, p2e_x, p1e_y, p2e_y, p3e_x, p3e_y;
int point1e_done = 0;
voidmyMouseFunc(intbutton, intstate, intx, inty)
{
        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;
        }
        elseif (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;
        }
        elseif (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));
```

```
                ellipsebres();
                point1e_done = 0;
        }
}
voidmyDrawing()
{ }
voidmyDrawingc()
{ }
Voidminit()
{
        glClearColor(1, 1, 1, 1);                                                    //Clears the
color (Fills the white color in background)
        glColor3f(1.0, 0.0, 0.0);                                                    //Sets the
Color to Red
        glPointSize(3.0);
        //Sets the display of the graphic to 3.0
        gluOrtho2D(-250, 250, -250, 250);                                    //Sets the window
position
}
voidmain(intargc, char* argv[])
{
        glutInit(&argc, argv);                                                       //Initialise
Glut Window
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);                 //Specifies Display Mode
        glutInitWindowSize(500, 500);                                        //Initialise Window
size
        glutInitWindowPosition(0, 0);                                        //Specifies Initial
Window position

        //FOR KEYBOARD
        printf("Enter 1 to draw circle , 2 to draw ellipse\n");
        intch;
        scanf_s("%d", &ch);
        switch (ch) {
        case 1:
                printf("Enter coordinates of centre of circle and radius\n");
                scanf_s("%d%d%d", &xc, &yc, &r);
                glutCreateWindow("Circle");
                                        //Creates window with title "Circle"
                glutDisplayFunc(circlebres);
                                //Executes function circlebres
                break;
        case 2:
                printf("Enter coordinates of centre of ellipse and major and minor radius\n");
                scanf_s("%d%d%d%d", &xce, &yce, &rx, &ry);
                glutCreateWindow("Ellipse");
                                //Creates window with title "Ellipse"
                glutDisplayFunc(ellipsebres);
                                //Executes function ellipsebres
                break;
        }
        //END KEYBOARD
        minit();
                                                //Initialise Window colors and buffer bit
        glutMainLoop();
                                                //Refreshes window
}
```

output




Enter 1 to draw circle , 2 to draw ellipse
2
Enter coordinates of centre of ellipse and major and minor radius
100
175
70
40
Ellipse

PROGRAM - 3

a. Write a program to recursively Subdivides a tetrahedron to form 3D Sierpinski gasket. The number of recursive steps is 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-traingle(point a, point b, Point C, int m);
Void draw-traingle(point p1, point p2, Point 3);
int main(int argv, char** argc)
{
    //int m;
    Point f("Enter the number of iterations:");
    Scan f-s("%d", &m);
    glut Init(&argv, argc);
    glut Init display mode(GLUT-SINGLE /GLUT-RGB |
GLUT-DEPTH);
    glut Init window position(100,200);
    glut Init window Size(500.500);
    glut creat window("Seirpinski Gasket");
```

Teacher's Signature _____

```
    glutDisplayFunc(tetrahedron);
    glEnable(GL_DEPTH_TEST);
    Myinit();
    glutMainLoop();
}
void divide_traingle(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;
    }
    else
        draw_traingle(a,b,c);
}
void myinit()
{
    glclearcolor(1,1,1,1);

    // glFlush();
    glortho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);
    //gluortho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);
}
void tetrahedron(void)
```

Teacher's Signature

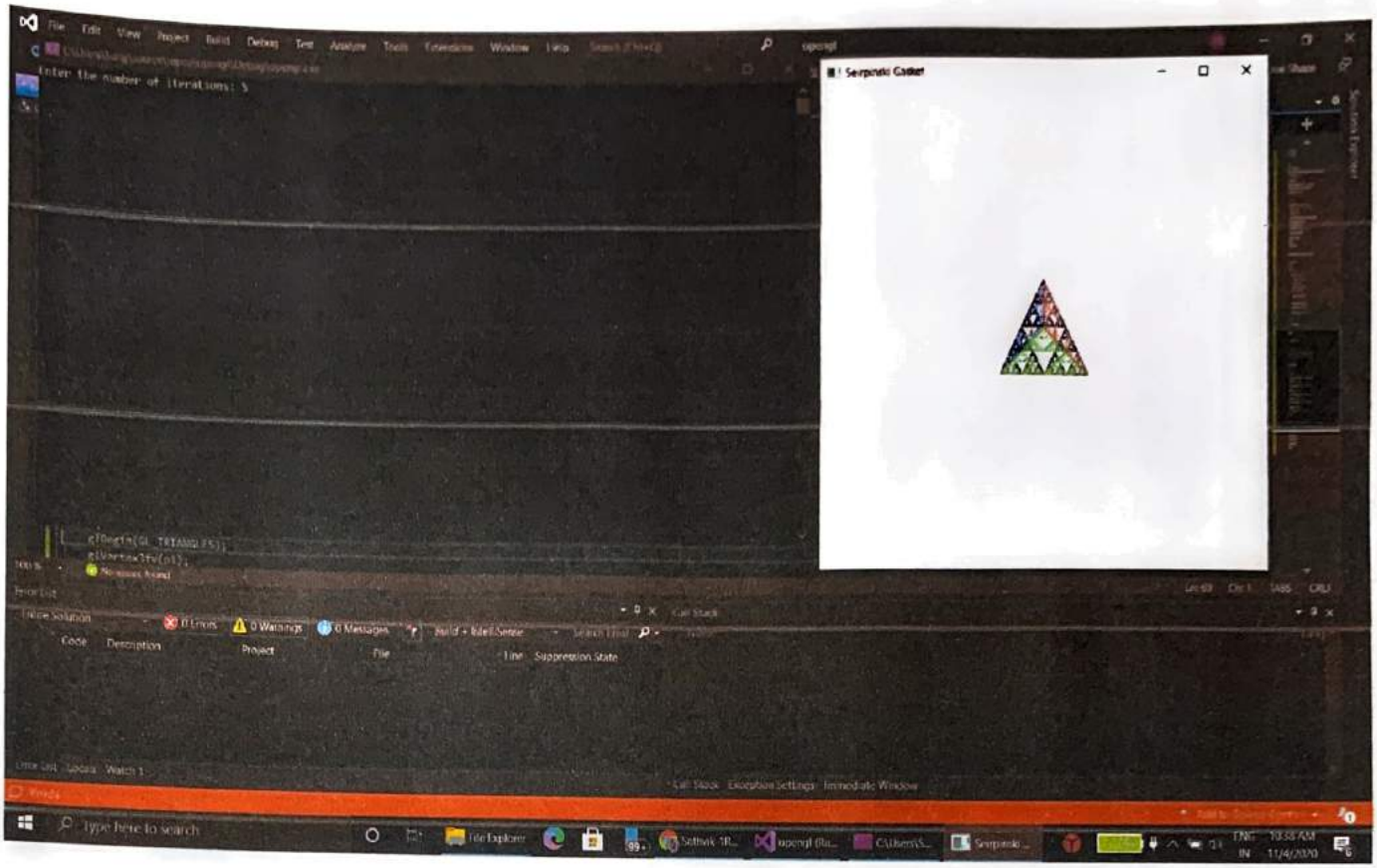
```
{
    //myinit();
    glclear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glcolor3f(1.0, 0.0, 0.0);
    divide_triangle(tetra[0], tetra[1], tetra[2], m);
    glcolor3f(0.0, 1.0, 0.0);
    divide_traingle(tetra[3], tetra[2], tetra[0], m);
    glcolor3f(0.0, 0.0, 1.0);
    divide_traingle(tetra[0], tetra[3], tetra[1], m);
    glcolor3f(0.0, 0.0, 0.0);
    divide_traingle(tetra[0], tetra[2], tetra[3], m);
    glFlush();
}
void draw_traingle(point p1, point p2, point p3)
{
    glBegin(GL_TRIANGLES);
    glvertex3fv(p1);
    glvertex3fv(p2);
    glvertex3fv(p3);
    glEnd();
}
```

program 3

```
#include<GL\glew.h>
#include<GL\freeglut.h>
#include<iostream>
//#define M 5
typedeffloatpoint[3];
pointtetra[4] = {
        {0,10,-10},
        {0,0,10},
        {10,-10,-10},
        {-10,-10,-10}
};
int M;

voiddraw_tetra(pointa, pointb, pointc, pointd) {
        glColor3f(0.0, 0.0, 0.0);
        draw_triangle(a, b, c);
        glColor3f(1.0, 0.0, 0.0);
        draw_triangle(a, c, d);
        glColor3f(0.0, 1.0, 0.0);
        draw_triangle(a, b, d);
        glColor3f(0.0, 0.0, 1.0);
        draw_triangle(b, c, d);
}

voiddraw_triangle(pointa, pointb, pointc) {
        glBegin(GL_TRIANGLES);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
        glEnd();
}

voiddivide_tetra(pointa, pointb, pointc, pointd, inti) {
        point v1, v2, v3, v4, v5, v6;
        if (i> 0) {
                for (int j = 0; j < 3; j++) {
                        v1[j] = (a[j] + b[j]) / 2;
                        v2[j] = (a[j] + c[j]) / 2;
                        v3[j] = (a[j] + d[j]) / 2;
                        v4[j] = (b[j] + c[j]) / 2;
                        v5[j] = (b[j] + d[j]) / 2;
                        v6[j] = (c[j] + d[j]) / 2;

                }
                divide_tetra(a, v1, v2, v3, i - 1);
                divide_tetra(v1, b, v4, v5, i - 1);
                divide_tetra(v2, v4, c, v6, i - 1);
                divide_tetra(v3, v5, v6, d, i - 1);

        }
        else
                draw_tetra(a, b, c, d);
}

voidtetrahedron() {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        //glBegin(GL_TRIANGLES);
        divide_tetra(tetra[0], tetra[1], tetra[2], tetra[3], M);
        //glEnd();

        glFlush();
}

voidmyinit() {
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1, 1, 1, 1);
        glOrtho(-12.0, 12.0, -12.0, 12.0, -12.0, 12.0);
}
intmain(intargv, char** argc)
{
        std::cout<<"Enter the number of iterations: ";
        std::cin>> M;
        glutInit(&argv, argc);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(900, 900);
        glutCreateWindow("Seirpinski Gasket");
        glutDisplayFunc(tetrahedron);
```

```
        glEnable(GL_DEPTH_TEST);
        myinit();
        glutMainLoop();
}
```

output



PROGRAM-4

2 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>

using namespace std;
float x[100], y[100]; //= {0,0,20,100,100},
 y[] = {0,100,50,100,0};

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) {
```

Teacher's Signature _____

```
float temp;
if (y2 < y1) {

    temp = x1, x1 = x2; x2 = temp;
    temp = y1, y1 = y2; y2 = temp;

}
if (scanline > y1 && scanline < y2)
    int x[m++] = x1 + (scanline - y1) *
        (x2 - x1) / (y2 - y1);
}
void Scanfill (float x[], float y[]) {
    for (int s1 = 0; s1 <= wy; s1++) {
        m = 0;
        for (int i = 0; i < n; i++) {
            edge detect (x[i], y[i], x[(i+1) % n],
                y[(i+1) % n], s1);
        }
        sort (intx, (intx + m));
        if (m >= 2)
            for (int i = 0; i < m; i = i + 2)
                draw_line (intx[i], s1, intx[i+1], s1);
    }
}
void display_filled_polygon() {
    g1 clear (GL_COLOR_BUFFER_BIT);
    g1 Linewidth (2);
    g1 Begin (GL_LINE_Loop);
    for (int i = 0; i < n; i++)
```

Teacher's Signature _____

```
            gl Vertex 2f (x[i], y[i]);
        gl End();
        Scan fill (x,y);
        //gl Flush();
}
Void my Init() {

        gl clear color (1.1,1,1);
        gl Color 3f (0.0,1);
        gl point Size (1);

        gl uuortho 2D (0, wx, 0, wy);

}
Void main (int ac, char* av[]) {
        glut Init (&ac, av);
        Print f ("Enter no, of sides: \n");
        Scan f ("%d", &n);
        Print f ("Enter coordinates of end points: \n");
        for (int i=0; i<n; i++)
        {
            print f("X-coord Y-coord: \n");
            Scan f ("%f %f", &x[i], &y[i]);
        }
        gl ut Init Display mode (GLUT_SINGLE | GLUT_RGB);
        gl ut Init window Size (500, 500);
        glut Init Windaw position (0,0);
        glut Creat Window ("Scan line");
        glut Display Func (display_filled_polygon);
        my Init();
        glut Main Loop();
}
```

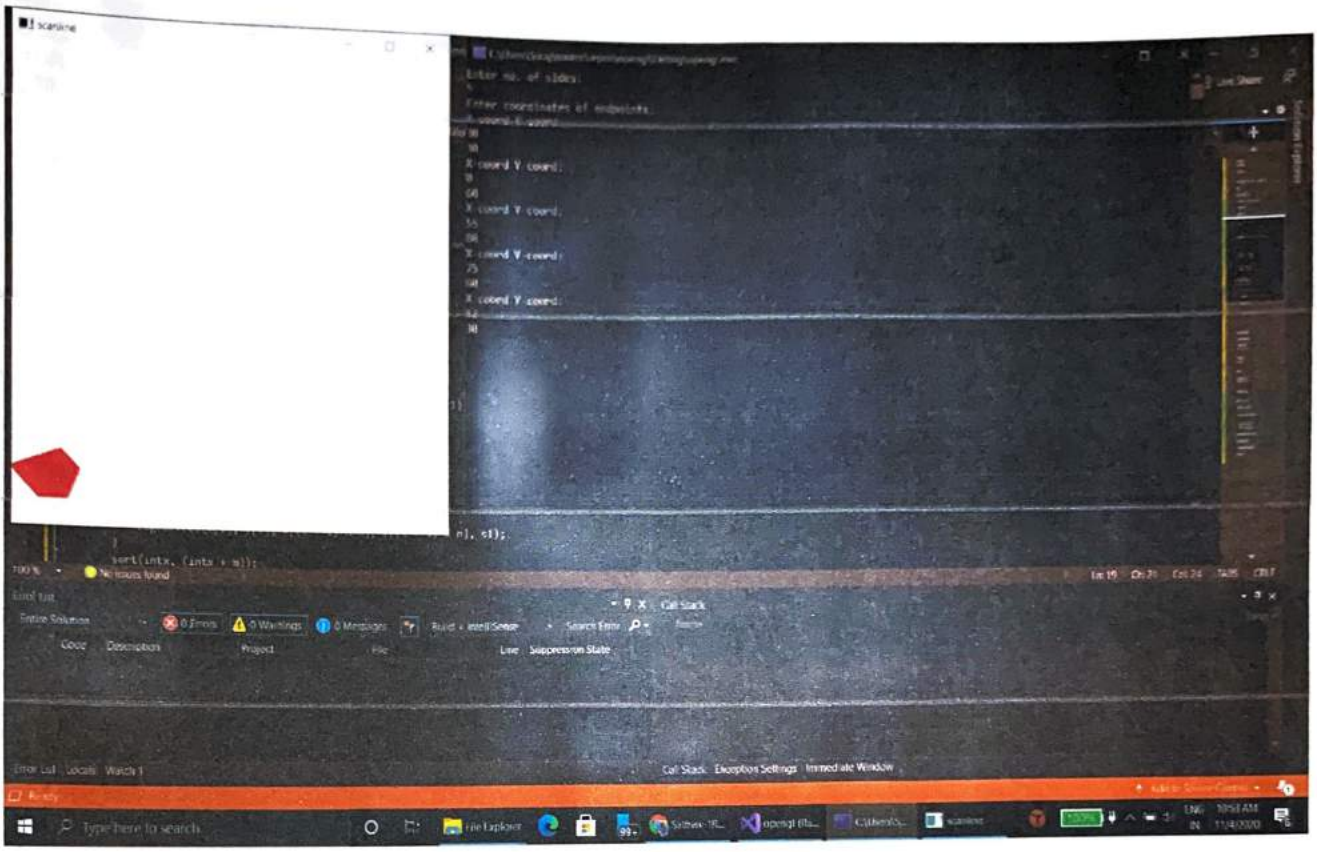
Teacher's Signature _____

program 4

```
#include<stdlib.h>
#include<gl/glut.h>
#include<algorithm>
#include<iostream>
#include<windows.h>
usingnamespace std;
floatx[100], y[100];
int n, m;
intwx = 500, wy = 500;
staticfloatintx[10] = { 0 };
voiddraw_line(floatx1, floaty1, floatx2, floaty2) {
        Sleep(100);
        glColor3f(1, 0, 0);
        glBegin(GL_LINES);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
        glEnd();
        glFlush();

}
voidedgeDetect(floatx1, floaty1, floatx2, floaty2, intscanline) {
        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);
}
voidscanfill(floatx[], floaty[]) {
        for (int s1 = 0; s1 <= wy; s1++) {
                m = 0;
                for (inti = 0; i< n; i++) {

                        edgeDetect(x[i], y[i], x[(i + 1) % n], y[(i + 1) % n], s1);
                }
                sort(intx, (intx + m));
                if (m >= 2)
                        for (inti = 0; i< m; i = i + 2)
                                draw_line(intx[i], s1, intx[i + 1], s1);
        }

}
voiddisplay_filled_polygon() {
        glClear(GL_COLOR_BUFFER_BIT);
        glLineWidth(2);
        glBegin(GL_LINE_LOOP);
        for (inti = 0; i< n; i++)
                glVertex2f(x[i], y[i]);
        glEnd();
        scanfill(x, y);
}
voidmyInit() {
        glClearColor(1, 1, 1, 1);
        glColor3f(0, 0, 1);
        glPointSize(1);
        gluOrtho2D(0, wx, 0, wy);

}
voidmain(intac, char* av[]) {
        glutInit(&ac, av);
        printf("Enter no. of sides: \n");
        scanf_s("%d", &n);
        printf("Enter coordinates of endpoints: \n");
        for (inti = 0; i< n; i++)
        {
                printf("X-coord Y-coord: \n");
                scanf_s("%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


scanline
Enter no. of sides:
Enter coordinates of endpoints:
X coord Y coord:
X coord Y coord:
X coord Y coord:
X coord Y coord:
sort(intx, intx + n);
No issues found
Call Stack
Entire Solution
0 Errors
0 Warnings
0 Messages
Build + IntelliSense
Search Error
Code
Description
Project
File
Line
Suppression State
Call Stack
Exception Settings
Immediate Window
Ready
Type here to search
File Explorer

PROGRAM - 5

a Write a program to create a house like figure and perform the following operations.
i. Rotate it about a given fixed point using open GL. transformation functions.
ii. Reflect it about an axis $y = mx + c$ using open GL transformation functions.

```
#include <gl/glut.h>
#include <math.h>
//#include <stdlib.h>
#include <stdio.h>

//RIGHT CLICK TO SHOW REFLECTED HOUSE

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();
```

Teacher's Signature _____

```
//NORMAL HOUSE
gl color3f (1,0,0);
gl Begin (GL_LINE_Loop);
for(int i=0; i<11; i++)
        gl vertex 2fv House[i]);
gl End();
gl Flush();
// ROTATED HOUSE
gl pushmatrix();
gl Translatef(100.100.0);
gl color3f (1. 1,0);
gl Begin(GL_LINE_Loop);
for (int i=0; i<11; i++)
        gl vertex 2fv House[i]);
gl End();
gl pop matrix();
gl Flush();
}
void display2()
{
    gl clear color(1,1,1,0);
    gl clear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    gl Matrix Mode (GL_PROJECTION);
    gl Load Identity();
    glu ortho2D (-450, 450, -450, 450);
    gl Matrix Mode (GL_MODELVIEW);
    gl Load Identity();
    // normal house
    gl color3f (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();
glTranslatef(0,c,0);
theta=atan(m);
theta=theta*180/3.14;
glRotatef(theta,0,0,1);
glScalef(1,-1,1);
glTranslatef(0,-c,0);
glRotatef(-theta,0,0,1);
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);
        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(); }
}
void main(int argc, char** argv)
{
    printf("Enter the rotation angle\n");
    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);
```

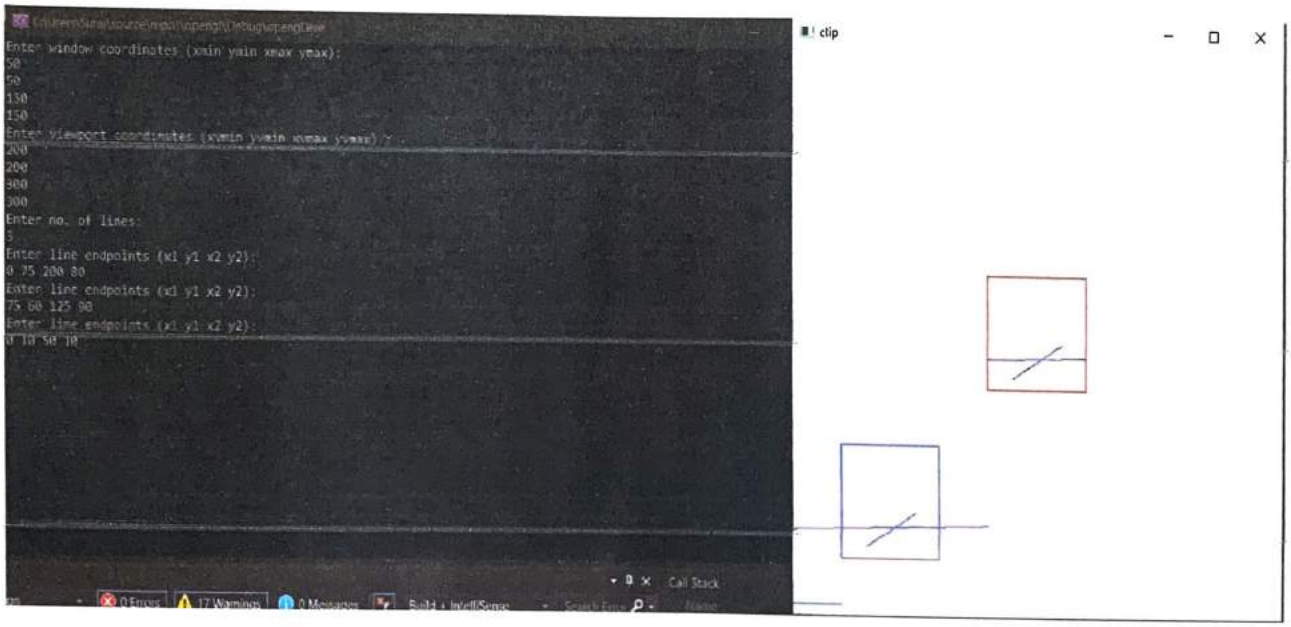
```
    glutInitWindowPosition(100,100);
    glutCreateWindow("House Rotation");
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    MyInit();
    glutMainLoop();
}
```

program 5

```
#include<gl/glut.h>
#include<math.h>
//#include<stdlib.h>
#include<stdio.h>
//RIGHT CLICK TO SHOW REFLECTED HOUSE
floathouse[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;

voiddisplay()
{
	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 (inti = 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 (inti = 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 (inti = 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();
	glTranslatef(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 (inti = 0; i< 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glPopMatrix();
        glFlush();
}
voidmyInit() {
        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);
}
voidmouse(intbtn, intstate, intx, inty) {
        if (btn == GLUT_LEFT_BUTTON &&state == GLUT_DOWN) {
                display();
        }
        elseif (btn == GLUT_RIGHT_BUTTON &&state == GLUT_DOWN) {
                display2();
        }
}
voidmain(intargc, char** argv)
{
        printf("Enter the rotation angle\n");
        scanf_s("%d", &angle);
        printf_s("Enter c and m value for line y=mx+c\n");
        scanf_s("%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



PROGRAM - 6

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

```
# define outcode int
# define true 1
# define false 0
double xmin, ymin, xmas, ymax;
double xvmin, yvmin, xvmax, yvmax;

const int RIGHT = 4;
const int LEFT = 8;
const int Top = 1;
const int Bottom = 2;

int n;
Struct line_Segment {
        int x1;
        int y1;
        int x2;
        int y2;
};
Struct line_Segment ls[10];

  out code computeoutcode (double x, double y)
{
```

```
outcode code = 0;
if (y > ymax)
        code |= TOP;
else if (y < ymin)
        code |= BOTTOM;
if (x > xmax)
        code |= RIGHT;
else if (x < xmin)
        code |= LEFT;
return code;
}
void cohenSutherland(double x0, double y0, double x1,
                     double y1)
{
outcode outcode0, outcode1, outcodeout;
bool accept = false, done = false;

outcode0 = ComputeOutcode(x0, y0);
outcode1 = ComputeOutcode(x1, y1);
do
{
    if (!(outcode0 | outcode1))
    {
        accept = true;
        done = true;
    }
    else if (outcode0 & outcode1)
        done = true;
    else
    {
```

Teacher's Signature _____

```
double x, y;
outcode out = outcode0 ? outcode0 : outcode1;
if (outcode out & TOP)
{
    x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
    y = ymax;
}
else if (outcode out & BOTTOM)
{
    x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
    y = ymin;
}
else if (outcode out & RIGHT)
{
    y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
    x = xmax;
}
else
{
    y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
    x = xmin;
}
if (outcode out == outcode0)
{
    x0 = x;
    y0 = y;
    outcode0 = computeoutcode (x0, y0);
}
else
{
```

Teacher's Signature _____

```
            x1 = x;
            y1 = y;
            outcode1 = computeOutCode(x1, y1);
        }
    }
} while(!done);
if (accept)
{
    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, 1);
    glBegin(GL_LINES);
    glVertex2d(vx0, vy0);
    glEnd();
}
}
void display()
```

Teacher's Signature _____

```
{
    gl clear (GL_COLOR_BUFFER_BIT);
    gl color 3f (0,0,1);
    gl Begin (GL_LINE_LOOP);
    gl vertex 2f (xmin, ymin);
    gl vertex 2f (xmax, ymin);
    gl vertex 2f (xmax, ymax);
    gl vertex 2f (xmin, ymax);
    gl End();
    for (int i=0; i<n; i++)
    {
        gl Begin (GL_LINES);
        gl vertex 2d (ls[i].x1, ls[i].y1);
        gl vertex 2d (ls[i].x2, ls[i].y2);
        gl End();
    }
    for (int i=0; i<n; i++)
        cohensutherl (ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);
    gl Flush();
}
void myinit();
{
    gl clear color (1.1.1.1);
    gl color 3f (1, 0, 0);
    gl point size (1.0);
    gl matrix Mode(GL_PROJECTION);
    gl load Identity();
    glu ortho 2D (0, 500, 0, 500);
}
```

Teacher's Signature _____

```
void main (int argc, char **argv)
{
    Printf ("Enter window coordinates (xmin ymin xmax ymax):\n");
    Scanf ("%lf %lf %lf %lf", &xmin, &ymin, &xmax, &ymax);
    Printf ("Enter viewport coordinates (xvmin yvmin xvmax yvmax):\n");
    Scanf ("%lf %lf %lf %lf", &xvmin, &yvmin, &xvmax, &yvmax);
    Printf ("Enter no. of lines:\n");
    Scanf ("%d", &n);
    for (int i=0; i<n; i++)
    {
        printf ("Enter line endpoints (x1 y1 x2 y2):\n");
        Scanf ("%d %d %d %d", &ls[i].x1, &ls[i].y1, &ls[i].x2,
                                &ls[i].y2);
    }
    glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (0, 0);
    glutCreatWindow ("Clip");
    MyInit ();
    glutDisplayFunc (display);
    glutMainLoop ();
}
```

Teacher's Signature _____

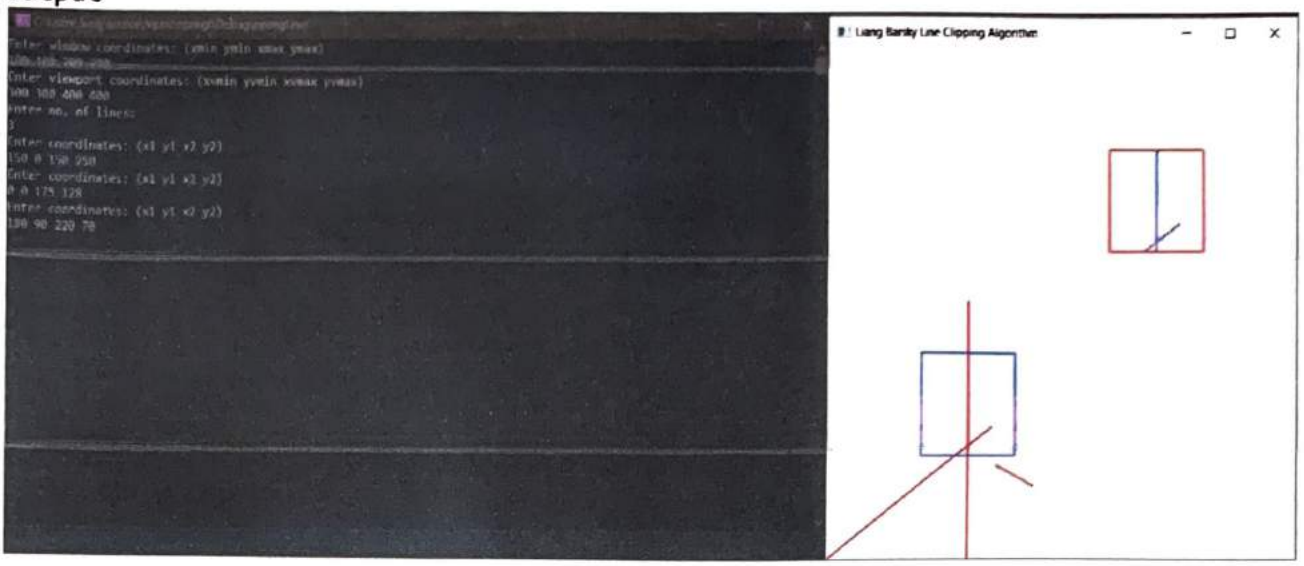
program 6

```
#include<stdio.h>
#include<stdlib.h>
#include<gl/glut.h>
#defineoutcodeint
#definetrue 1
#definefalse 0
doublexmin, ymin, xmax, ymax;
doublexvmin, yvmin, xvmax, yvmax;
constint RIGHT = 4;
constint LEFT = 8;
constint TOP = 1;
constint BOTTOM = 2;
int n;
structline_segment {
        int x1;
        int y1;
        int x2;
        int y2;
};
structline_segmentls[10];
outcodecomputeoutcode(doublex, doubley)
{
        outcode code = 0;
        if (y>ymax)
                code |= TOP;
        elseif (y<ymin)
                code |= BOTTOM;
        if (x>xmax)
                code |= RIGHT;
        elseif (x<xmin)
                code |= LEFT;
        return code;
}
voidcohensuther(doublex0, doubley0, doublex1, doubley1)
{
        outcode outcode0, outcode1, outcodeout;
        bool accept = false, done = false;
        outcode0 = computeoutcode(x0, y0);
        outcode1 = computeoutcode(x1, y1);
        do
        {
                if(!(outcode0 | outcode1))
                {
                        accept = true;
                        done = true;
                }
                elseif (outcode0 & outcode1)
                        done = true;
                else
                {
                        double x, y;
                        outcodeout = outcode0 ? outcode0 : outcode1;
                        if (outcodeout& TOP)
                        {
                                x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
                                y = ymax;
                        }
                        elseif (outcodeout& BOTTOM)
                        {
                                x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
                                y = ymin;
                        }
                        elseif (outcodeout& RIGHT)
                        {
                                y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
                                x = xmax;
                        }
                        else
                        {
                                y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
                                x = xmin;
                        }

                        if (outcodeout == outcode0)
                        {
                                x0 = x;
                                y0 = y;
                                outcode0 = computeoutcode(x0, y0);
```

```
                }
                else
                {
                        x1 = x;
                        y1 = y;
                        outcode1 = computeoutcode(x1, y1);
                }
        }

    } while(!done);
    if (accept)
    {
            doublesx = (xvmax - xvmin) / (xmax - xmin);
            doublesy = (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);
            glBegin(GL_LINE_LOOP);
            glVertex2f(xvmin, yvmin);
            glVertex2f(xvmax, yvmin);
            glVertex2f(xvmax, yvmax);
            glVertex2f(xvmin, yvmax);
            glEnd();
            glColor3f(0, 0, 1);
            glBegin(GL_LINES);
            glVertex2d(vx0, vy0);
            glVertex2d(vx1, vy1);
            glEnd();
    }
}
voiddisplay()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0, 0, 1);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xmin, ymin);
    glVertex2f(xmax, ymin);
    glVertex2f(xmax, ymax);
    glVertex2f(xmin, ymax);
    glEnd();
    for (inti = 0; i< n; i++)
    {
            glBegin(GL_LINES);
            glVertex2d(ls[i].x1, ls[i].y1);
            glVertex2d(ls[i].x2, ls[i].y2);
            glEnd();}
    for (inti = 0; i< n; i++)
            cohensuther(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);

    glFlush();
}
voidmyinit()
{
    glClearColor(1, 1, 1, 1);
    glColor3f(1, 0, 0);
    glPointSize(1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0, 500, 0, 500);
}

voidmain(intargc, char** argv)
{
    printf("Enter window coordinates (xminyminxmaxymax): \n");
    scanf_s("%lf%lf%lf%lf", &xmin, &ymin, &xmax, &ymax);
    printf("Enter viewport coordinates (xvminyvminxvmaxyvmax) :\n");
    scanf_s("%lf%lf%lf%lf", &xvmin, &yvmin, &xvmax, &yvmax);
    printf("Enter no. of lines:\n");
    scanf_s("%d", &n);
    for (inti = 0; i< n; i++)
    {
        printf("Enter line endpoints (x1 y1 x2 y2):\n");
        scanf_s("%d%d%d%d", &ls[i].x1, &ls[i].y1, &ls[i].x2, &ls[i].y2);
    }
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("clip");
    myinit();
```

```
        glutDisplayFunc(display);
        glutMainLoop();
}
```

output



PROGRAM. 7

Q Write a program to implement the Liang-Barsky line clipping algorithm. Make provision to specify the input for multiple lines, window for clipping and viewport for displaying the clipped Image.

```
#include <stdio.h>
#include <GL/glut.h>

double xmin, ymin, xmax, ymax; //50 50 100 100
double xvmin, yvmin, xvmax, yvmax; //200 200 300 300
int n;

struct line_segment {
        int x1;
        int y1;
        int x2;
        int y2;
};
struct line_segment ls[10];
int cliptest(double p, double q, double *u1, double *
{
        double r;
        if(p) r=q/p; // to check whether p
        if (p<0.0) // potentially entry point, update te
        {
                if (r>*u1) *u1=r;
                if (r>*u2) return (false); // line portion is outside
```

```
}
else
        if(p>0.0) // potentially leaving point, update t1
        {
            if(r<*u2)*u2=r;
            if(r<*u1) 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);
}
Void LiangBarSkylineClipandDraw(double x0, double y0,
                                double x1, double y1)
{
    double dx = x1-x0, dy=y1-y0, u1=0.0, u2=1.0;
    // draw a red colored viewport.
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_LINE_Loop);
    glVertex2f(xvmin, yvmin);
    glVertex2f(xvmax, yvmax);
    glVertex2f(xvmax, yvmax);
    glVertex2f(xvmin, yvmax);
    glEnd();
    if(Cliptest(-dx, x0-xmin, &u1, &u2)) // inside
                                          test wrt left edge.
```

Teacher's Signature _____

```
if (cliptest(dx, xmax, -x0, &u1, &u2)) // inside test
                                    wrt right edge
  if (cliptest(-dy, y0-ymin, &u1, &u2)) // inside
                                    test wrt bottom edge.
    if (cliptest(dy, ymax, -y0, &u1, &u2)) // inside
                                    test wrt Top edge.
    {
      if (u2 < 1.0)
      {
        x1 = x0 + u2*dx;
        y1 = y0 + u2*dy;
      }
      // window to viewport mappings
      double Sx = (xvmax - xvmin)/(xmax - ymin); //
                    double Scale parameter
      double Sy = (yvmax - xvmin)/(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;

      gl color 3f (0.0, 0.0, 1.0); // draw blue colored
                                    clipped line
      gl Begin (GL_LINES);
      gl vertex 2d (Vx0, Vy0);
      gl Vertex 2d (Vx1, Vy1);
      gl End();
    }
```

Teacher's Signature _____

```
}// end of line clipping

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    // draw the line with red color
    glColor3f(1.0, 0.0, 0.0);
    for(int i=0; i<n; i++)
    {
        glBegin(GL_LINES);
        glVertex2d(ls[i].x1, ls[i].y1);
        glVertex2d(ls[i].x2, ls[i].y2);
        glEnd();
    }
    // draw a blue colored window
    glColor3f(0.0, 0.0, 1.0);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xmin, ymin);
    glVertex2f(xmax, ymin);
    glVertex2f(xmax, ymax);
    glVertex2f(xmin, ymax);
    glEnd();
    for(int i=0; i<n; i++)
        LiangBarSkyLineClipAndDraw(ls[i].x1, ls[i].y1,
                                   ls[i].x2, ls[i].y2);
    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(0.0, 499.0, 0.0, 499.0);
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);

    printf("Enter window coordinates:(xmin ymin xmax ymax)\n");
    scanf("%lf %lf %lf %lf", &xmin, &ymin, &xmax, &ymax);
    printf("Enter viewport coordinates:(xvmin yvmin xvmax yvmax)\n");
    scanf("%lf %lf %lf %lf", &xvmin, &yvmin, &xvmax, &yvmax);
    printf("Enter no. of lines:\n");
    scanf("%d", &n);

    for(int i=0; i<n; i++)
    {
        printf("Enter coordinates:(x1 y1 x2 y2)\n");
        scanf("%d%d%d%d", &ls[i].x1, &ls[i].y1, &ls[i].x2,
                                                &ls[i].y2);
    }
    glutCreateWindow("Liang Barsky line clipping Algorithm");
    glutDisplayFunc(display);
    myInit();
    glutMainLoop();
}
```

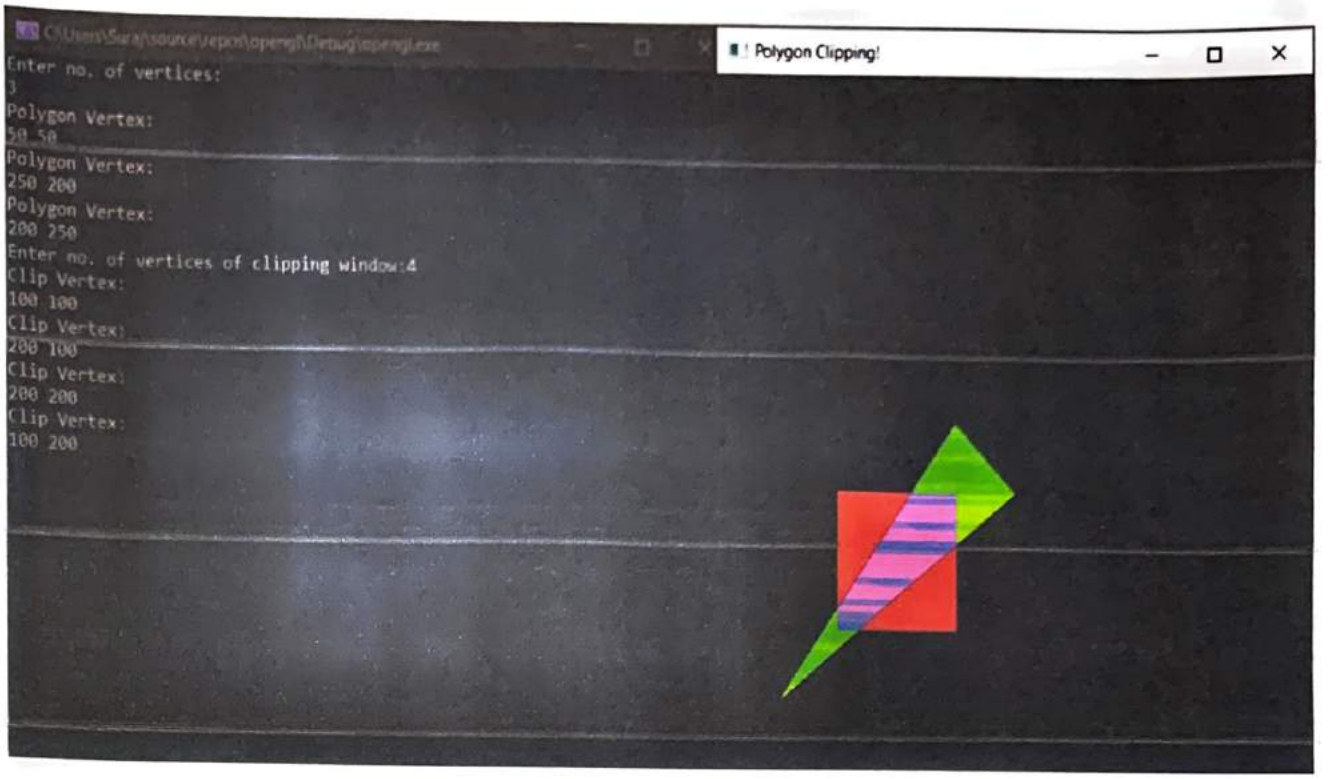
Teacher's Signature _____

program 7

```
#include<stdio.h>
#include<GL/glut.h>

doublexmin, ymin, xmax, ymax; //50 50 100 100
doublexvmin, yvmin, xvmax, yvmax; //200 200 300 300

int n;

structline_segment {
        int x1;
        int y1;
        int x2;
        int y2;
};
structline_segmentls[10];
intcliptest(doublep, doubleq, double* u1, double* u2)
{
        double r;
        if (p) r = q / p;  // to check whether p
        if (p< 0.0)    // potentially entry point, update te
        {
                if (r > * u1)*u1 = r;
                if (r > * u2) return(false); // line portion is outside
        }
        else
                if (p> 0.0)    //  Potentially leaving point, update tl
                {
                        if (r < *u2)*u2 = r;
                        if (r < *u1) 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);
}

voidLiangBarskyLineClipAndDraw(doublex0, doubley0, doublex1, doubley1)
{
        double dx = x1 - x0, dy = y1 - y0, u1 = 0.0, u2 = 1.0;
        //draw a red colored viewport
        glColor3f(1.0, 0.0, 0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(xvmin, yvmin);
        glVertex2f(xvmax, yvmin);
        glVertex2f(xvmax, yvmax);
        glVertex2f(xvmin, yvmax);
        glEnd();
        if (cliptest(-dx, x0 - xmin, &u1, &u2))  // inside test wrt left edge
                if (cliptest(dx, xmax - x0, &u1, &u2)) // inside test wrt right edge
                        if (cliptest(-dy, y0 - ymin, &u1, &u2)) // inside test wrt bottom edge
                                if (cliptest(dy, ymax - y0, &u1, &u2)) // inside test wrt top edge
                                {
                                        if (u2 < 1.0)
                                        {
                                                x1 = x0 + u2 * dx;
                                                y1 = y0 + u2 * dy;
                                        }
                                        if (u1 > 0.0)
                                        {
                                                x0 = x0 + u1 * dx;
                                                y0 = y0 + u1 * dy;
                                        }
                                        // Window to viewport mappings
                                        doublesx = (xvmax - xvmin) / (xmax - xmin); // Scale parameters
                                        doublesy = (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(0.0, 0.0, 1.0); // draw blue colored clipped line
                                        glBegin(GL_LINES);
                                        glVertex2d(vx0, vy0);
                                        glVertex2d(vx1, vy1);
                                        glEnd();
                                }
```

```
}// end of line clipping

voiddisplay()
{
        glClear(GL_COLOR_BUFFER_BIT);
        //draw the line with red color
        glColor3f(1.0, 0.0, 0.0);
        for (inti = 0; i< n; i++)
        {
                glBegin(GL_LINES);
                glVertex2d(ls[i].x1, ls[i].y1);
                glVertex2d(ls[i].x2, ls[i].y2);
                glEnd();
        }
        //draw a blue colored window
        glColor3f(0.0, 0.0, 1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(xmin, ymin);
        glVertex2f(xmax, ymin);
        glVertex2f(xmax, ymax);
        glVertex2f(xmin, ymax);
        glEnd();
        for (inti = 0; i< n; i++)
                LiangBarskyLineClipAndDraw(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);
        glFlush();
}

voidmyinit()
{
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(1.0, 0.0, 0.0);
        glLineWidth(2.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}

intmain(intargc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(0, 0);

        printf_s("Enter window coordinates: (xminyminxmaxymax) \n");
        scanf_s("%lf%lf%lf%lf", &xmin, &ymin, &xmax, &ymax);
        printf_s("Enter viewport coordinates: (xvminyvminxvmaxyvmax) \n");
        scanf_s("%lf%lf%lf%lf", &xvmin, &yvmin, &xvmax, &yvmax);
        printf_s("Enter no. of lines:\n");
        scanf_s("%d", &n);

        for (inti = 0; i< n; i++)
        {
                printf_s("Enter coordinates: (x1 y1 x2 y2)\n");
                scanf_s("%d%d%d%d", &ls[i].x1, &ls[i].y1, &ls[i].x2, &ls[i].y2);
        }
        glutCreateWindow("Liang Barsky Line Clipping Algorithm");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

output


Enter no. of vertices:
3
Polygon Vertex:
50 50
Polygon Vertex:
250 200
Polygon Vertex:
200 250
Enter no. of vertices of clipping window:4
Clip Vertex:
100 100
Clip Vertex:
200 100
Clip Vertex:
200 200
Clip Vertex:
100 200
Polygon Clipping!

PROGRAM-8

6. Write a program to implement the Cohen-Hodgeman polygon clipping algorithm. Make provision to specify the input polygon and window for clipping.

```
// C++ program for implementing Sutherland-Hodgman
// algorithm for polygon clipping.
#include <ioStream>
#include <GL/glut.h>
using namespace std;
int poly_size, poly_points[20][2], org_poly_size, or-
                                    g_poly_points[20][2],
clipper_size, clipper_points[20][2];
const int MAX_POINTS = 20;

// Returns x-value of point of intersection of two
// lines
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_intersect(int x1, int y1, int x2, int y2,
        int x3, int y3, int x4, int y4)
{
    int num = (x1*y2 - y1*x2)*(x3 - x4) -
              (x1 - x2)*(x3*y4 - y3*x4);
```

Teacher's Signature

```
    int den = (x1-x2)*(y3-y4)-(y1-y2)*(x3-x4);
    return num/den;
}
// Returns y-value of point of intersection of
// two lines
int y_intersect(int x1, int y1, int x2, int y2,
    int x3, int y3, int x4, int y4)
{
    int num = (x1*y2-y1*x2)*(y3-y4)-
              (y1-y2)*(x3*y4-y3*x4);
    int den = (x1-x2)*(y3-y4)-(y1-y2)*(x3-x4);
    return num/den;
}
// This functions clips all the edges w.r.t one clip
// edge of clipping area
void Clip(int poly_points[][2], int& poly_size,
    int x1, int y1, int x2, int y2)
{
    int new_points[MAX_POINTS][2], new_poly_size=0;
    // (ix, iy), (kx, ky) are the co-ordinate values of
    // the points
    for (int i=0; i<poly_size; i++)
    {
        // i and k form a line in polygon
        int k=(i+1)%poly_size;
        int ix=poly_points[i][0], iy=poly_points[i][1];
        int kx=poly_points[k][0], ky=poly_points[k][1];
```

```
// Calculating position of first point
// w.r.t. clipper line
int i_pos = (x2-x1)*(iy-y1) - (y2-y1)*(ix-x1);

// Calculating position of second point
// w.r.t. clipper line
int k_pos = (x2-x1)*(ky-y1) - (y2-y1)*(kx-x1);

// case 1 : when both points are inside
if (i_pos >= 0 && k_pos >= 0)
{
    // only second point is added
    new_points[new_poly_size][0] = kx;
    new_points[new_poly_size][1] = ky;
    new_poly_size++;
}

// case 2 : when only first point is outside
else if (i_pos < 0 && k_pos >= 0)
{
    // point of intersection with edge
    // and the second point is added
    new_points[new_poly_size][0] = x_intersect
            (x1, y1, x2, y2, ix, iy, kx, ky);
    new_points[new_poly_size][1] = y_intersect
            (x1, y1, x2, y2, ix, iy, kx, ky);
    new_poly_size++;

    new_points[new_poly_size][0] = kx;
    new_points[new_poly_size][1] = ky;
    new_poly_size++;
}
```

Teacher's Signature _____

```
// Case 3; When only Second point is out side
else if (i_pos >= 0 && k_pos < 0)
{

    // only point of intersection with edge is added
    new_points[new_poly_size][0] = x_intersect
        (x1, y1, x2, y2, ix, iy, kx, ky);
    new_points[new_poly_size][1] = y_intersect
        (x1, y1, x2, y2, ix, iy, kx, ky);
    new_poly_size++;
}
// case 4: when both points are outside
else
{
    // No points are added
}
}

// copying new points into original array
// and changing the no. of vertices.
poly_size = new_poly_size;
for (int i=0; i< poly_size; i++)
{
    poly_points[i][0] = new_points[i][0];
    poly_points[i][1] = new_points[i][1];
}
}

Void init() {
    glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
```

Teacher's Signature _____

```
    glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 500.0);
    glClear(GL_COLOR_BUFFER_BIT);
}
// Implement Sutherland's Hodgman algorithm
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);
    // i and k are two consecutive indexes

    for(int i=0; i<clipper_Size; i++)
    {
        int k=(i+1)%clipper_Size;

        // we pass the current array of vertices, it's size
        // and the end points of selected clipper line
        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();
}
```

Teacher's Signature _____

```
// Driver code
int main (int argc, char* argv[])
{
        printf("Enter no. of vertices :\n");
        scanf_s("%d", &poly_size);
        org_poly_size = poly_size;
        for (int i=0; i< poly_size; i++)
        {
            Printf("polygon vertex :\n");
            Scanf_s("%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];
        }
printf("Enter no. of vertices of clipping window:");
scanf_s("%d", &clipper_size);
for (int i=0; i< clipper_size; i++)
{
        printf("clip vertex:\n");
        scanf_s("%d%d", &clipper_points[i][0], &
                                clipper_points[i][1];
}

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

Teacher's Signature _____

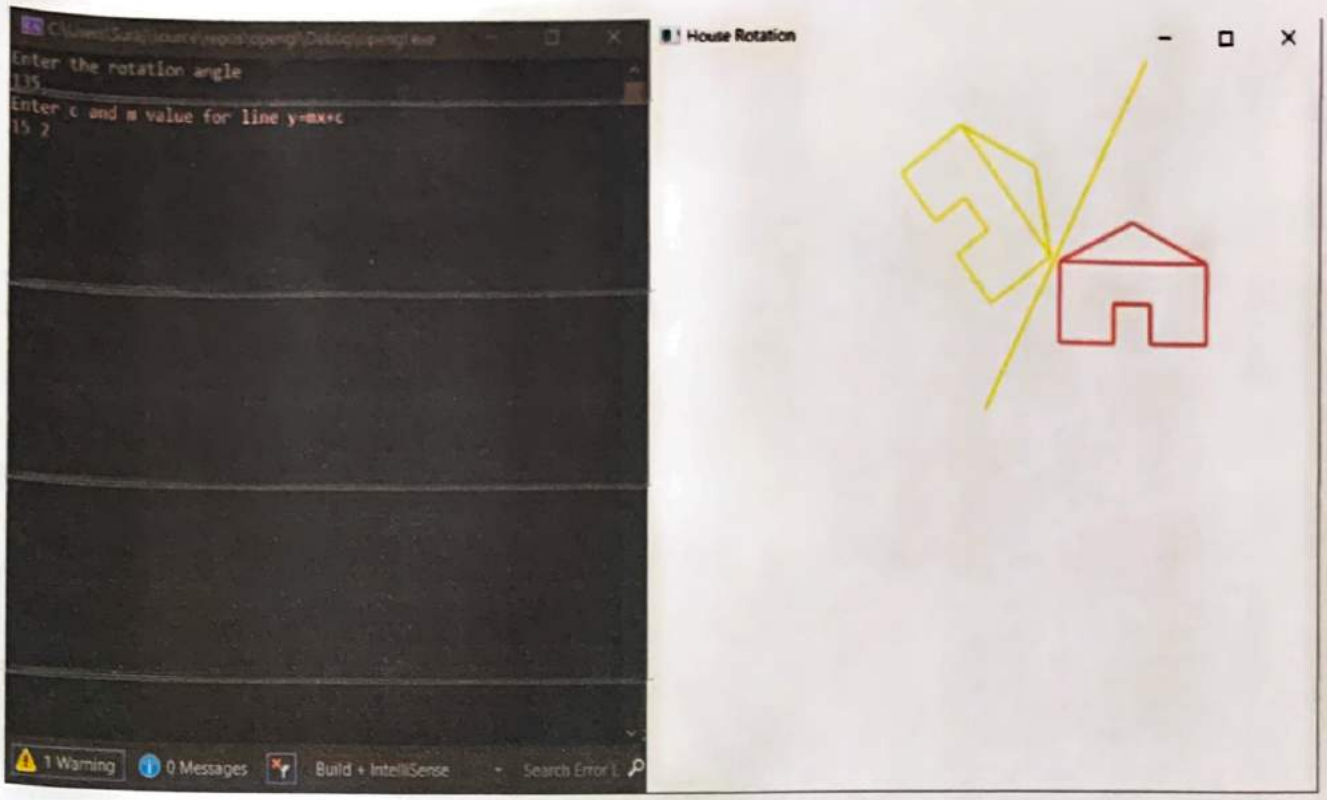
program 8

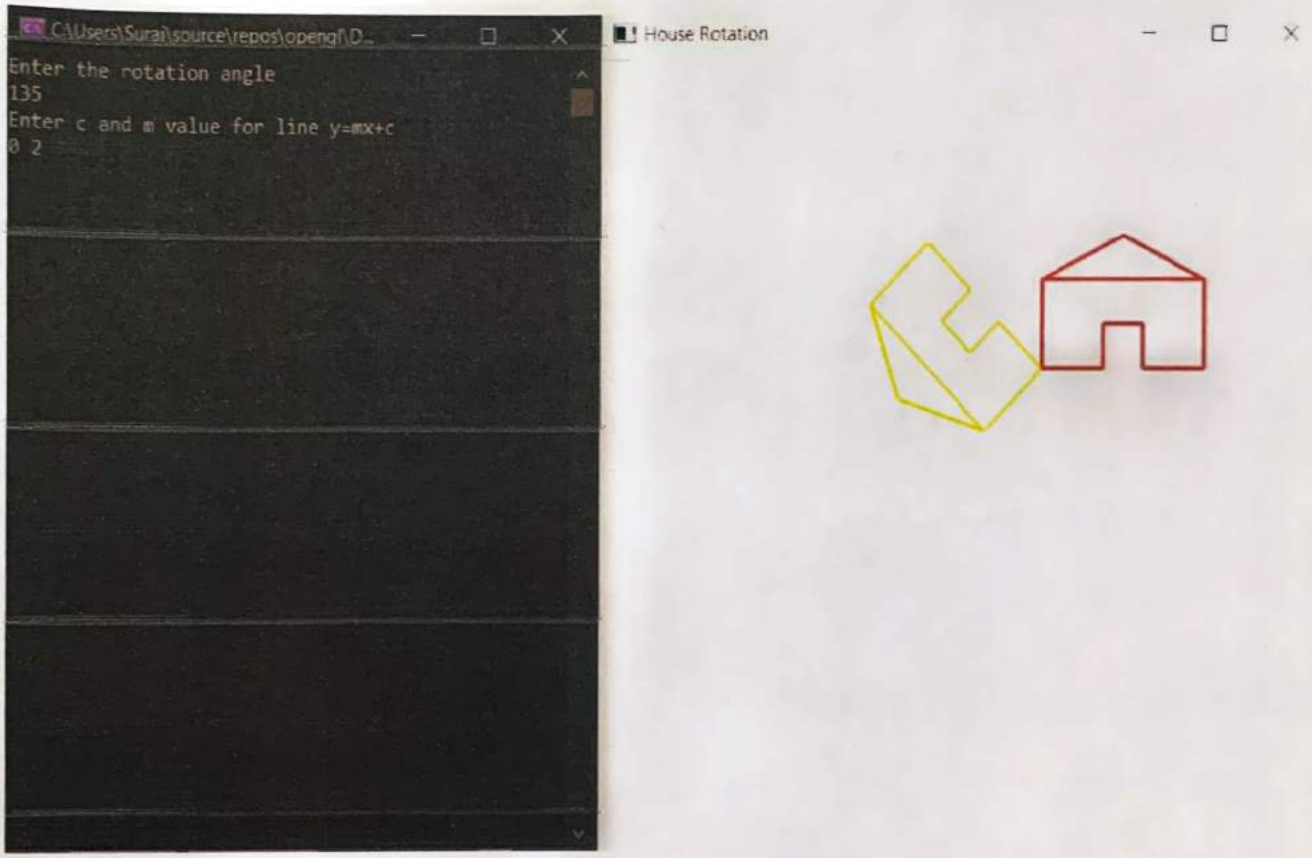
```
#include<iostream>
#include<GL/glut.h>
usingnamespace std;
intpoly_size, poly_points[20][2], org_poly_size, org_poly_points[20][2], clipper_size, clipper_points[20][2];
constint MAX_POINTS = 20;

// Returns x-value of point of intersection of two lines
voiddrawPoly(intp[][2], intn) {
        glBegin(GL_POLYGON);
        for (inti = 0; i<n; i++)
                glVertex2f(p[i][0], p[i][1]);
        glEnd();
}

intx_intersect(intx1, inty1, intx2, inty2,
        intx3, inty3, intx4, inty4)
{
        intnum = (x1 * y2 - y1 * x2) * (x3 - x4) -
                (x1 - x2) * (x3 * y4 - y3 * x4);
        int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
        returnnum / den;
}

// Returns y-value of point of intersectipn of two lines
inty_intersect(intx1, inty1, intx2, inty2,
        intx3, inty3, intx4, inty4)
{
        intnum = (x1 * y2 - y1 * x2) * (y3 - y4) -
                (y1 - y2) * (x3 * y4 - y3 * x4);
        int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
        returnnum / den;
}

// This functions clips all the edges w.r.t one clip
// edge of clipping area
voidclip(intpoly_points[][2], int&poly_size,
        intx1, inty1, intx2, inty2)
{
        intnew_points[MAX_POINTS][2], new_poly_size = 0;

        // (ix,iy),(kx,ky) are the co-ordinate values of
        // the points
        for (inti = 0; i<poly_size; i++)
        {
                // i and k form a line in polygon
                int k = (i + 1) % poly_size;
                int ix = poly_points[i][0], iy = poly_points[i][1];
                intkx = poly_points[k][0], ky = poly_points[k][1];

                // Calculating position of first point
                // w.r.t. clipper line
                inti_pos = (x2 - x1) * (iy - y1) - (y2 - y1) * (ix - x1);

                // Calculating position of second point
                // w.r.t. clipper line
                intk_pos = (x2 - x1) * (ky - y1) - (y2 - y1) * (kx - x1);

                // Case 1 : When both points are inside
                if (i_pos>= 0 &&k_pos>= 0)
                {
                        //Only second point is added
                        new_points[new_poly_size][0] = kx;
                        new_points[new_poly_size][1] = ky;
                        new_poly_size++;
                }

                // Case 2: When only first point is outside
                elseif (i_pos< 0 &&k_pos>= 0)
                {
                        // Point of intersection with edge
                        // and the second point is added
                        new_points[new_poly_size][0] = x_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_points[new_poly_size][1] = y_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_poly_size++;

                        new_points[new_poly_size][0] = kx;
                        new_points[new_poly_size][1] = ky;
```

```
                        new_poly_size++;
                }

                // Case 3: When only second point is outside
                elseif (i_pos>= 0 &&k_pos< 0)
                {
                        //Only point of intersection with edge is added
                        new_points[new_poly_size][0] = x_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_points[new_poly_size][1] = y_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_poly_size++;
                }
                // Case 4: When both points are outside
                else
                {}
        }
        poly_size = new_poly_size;
        for (inti = 0; i<poly_size; i++)
        {
                poly_points[i][0] = new_points[i][0];
                poly_points[i][1] = new_points[i][1];
        }
}
voidinit() {
        glClearColor(0.0f, 0.0f, 0.0f, 0.0f);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 500.0);
        glClear(GL_COLOR_BUFFER_BIT);
}
voiddisplay()
{
        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);
        //i and k are two consecutive indexes
        for (inti = 0; i<clipper_size; i++)
        {
                int k = (i + 1) % clipper_size;

                // We pass the current array of vertices, it's size
                // and the end points of the selected clipper line
                clip(poly_points, poly_size, clipper_points[i][0],
                        clipper_points[i][1], clipper_points[k][0],
                        clipper_points[k][1]);
        }
        glColor3f(0.0f, 0.0f, 1.0f);
        drawPoly(poly_points, poly_size);
        glFlush();
}
intmain(intargc, char* argv[])
{
        printf("Enter no. of vertices: \n");
        scanf_s("%d", &poly_size);
        org_poly_size = poly_size;
        for (inti = 0; i<poly_size; i++)
        {
                printf("Polygon Vertex:\n");
                scanf_s("%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];
        }

        printf("Enter no. of vertices of clipping window:");
        scanf_s("%d", &clipper_size);
        for (inti = 0; i<clipper_size; i++)
        {
                printf("Clip Vertex:\n");
                scanf_s("%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 the rotation angle
135
Enter c and m value for line y=mx+c
15 2
1 Warning
0 Messages
Build + IntelliSense
House Rotation


C:\Users\Sura\source\repos\opengl\D...
Enter the rotation angle
135
Enter c and m value for line y=mx+c
0 2
House Rotation

Date

Expt. No. Page No. 42

PROGRAM - 9

Write a program to model a car like figure using display lists and move a car from one end of the screen to other end. user is able to control 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(80,55,0);
        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);
```

Teacher's Signature

```
        glut solidsphere (10,25,25);
        gl EndList();
}
void MyKeyboard (unsigned char key, int x, int y) {
        Switch (key) {
        case 't': glut postRedisplay();
                break;
        case 'q': exit (0);
        default : break;
        }
}
void MyInit () {
        gl clearcolor (0, 0, 0, 0);
        gl ortho (0.600, 0.600, 0.600);
}
void draw_wheel () {
        gl color 3f (0,1,1);
        glut Solidsphere (10, 25, 25);
}
void move car (float s) {
        gl Translate f (s, 0.0, 0.0);
        gl call List (CAR);
        gl Push Matrix ();
        gl Translate (25, 25, 0.0); // move to first wheel position
        // draw-wheel();
        gl CallList (WHEEL);
        gl pop Matrix ();
        gl push Matrix ();
        gl Translate f (75, 25, 0.0); // move to second wheel position
```

Teacher's Signature _____

```
    ////draw_wheel();
    glCallList(WHEEL);
    glPopMatrix();
    glFlush();
}
void myDisp() {
    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) {
        s+=5;
        myDisp();
    }
    else if (btn == GLUT_RIGHT_BUTTON && state ==
        s+=2;                                GLUT_DOWN) {
        myDisp();
    }
}
int main(int argc, char* argv[]) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 500);
    glutInitWindowPosition(100, 100);
    glutCreatWindow("car");
    myInit();
    glutDisplayFunc(myDisp);
    glutMouseFunc(mouse);
    glutKeyboardFunc(mykeyboard);
    glutMainLoop();
}
```

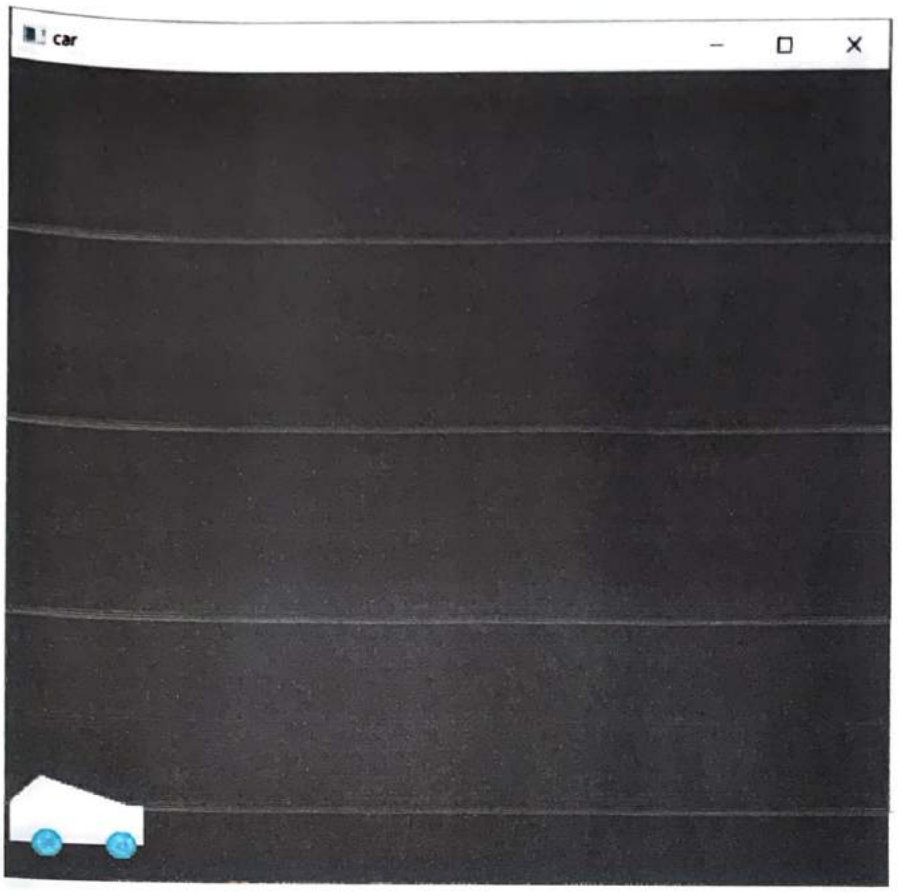
Teacher's Signature _____

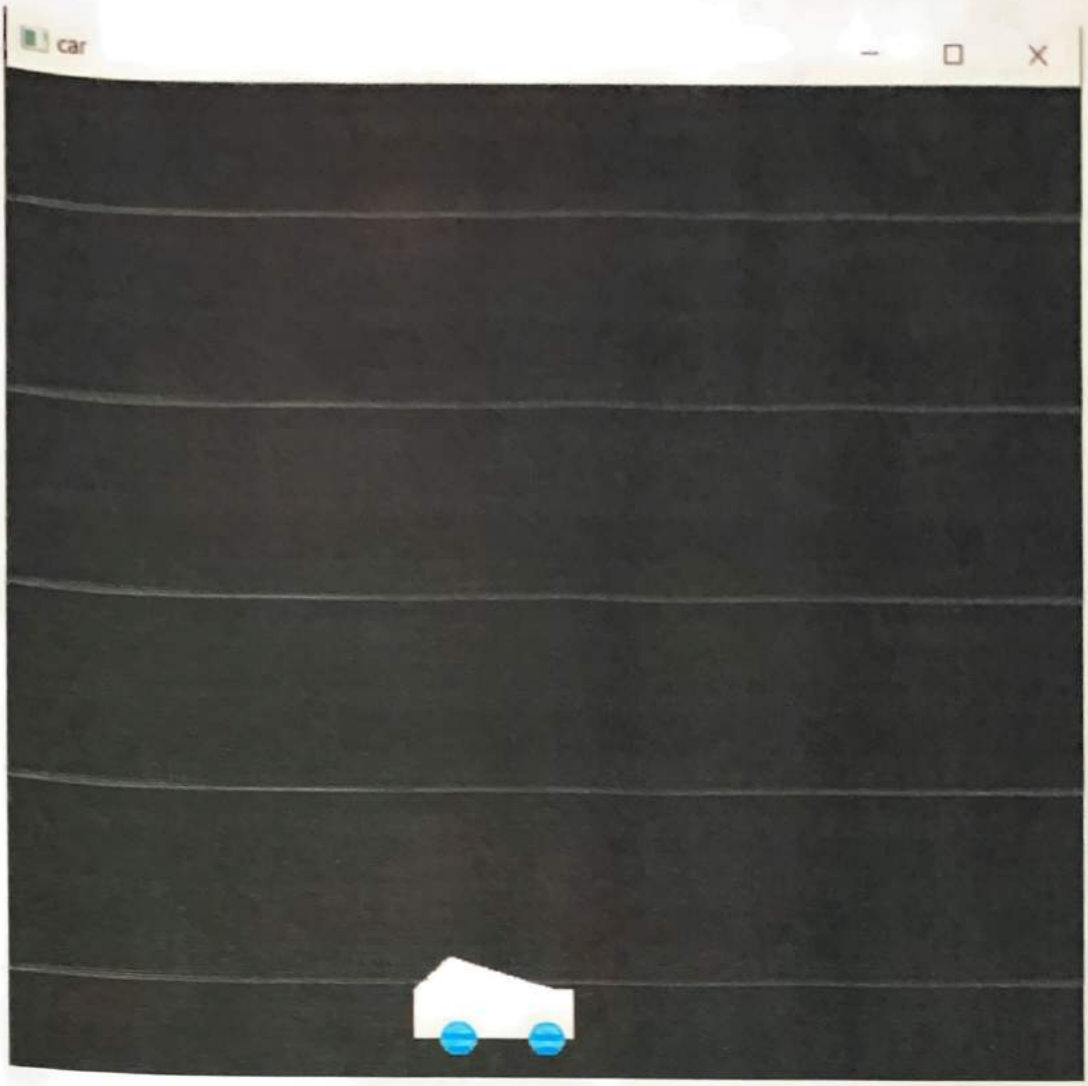
Program 9

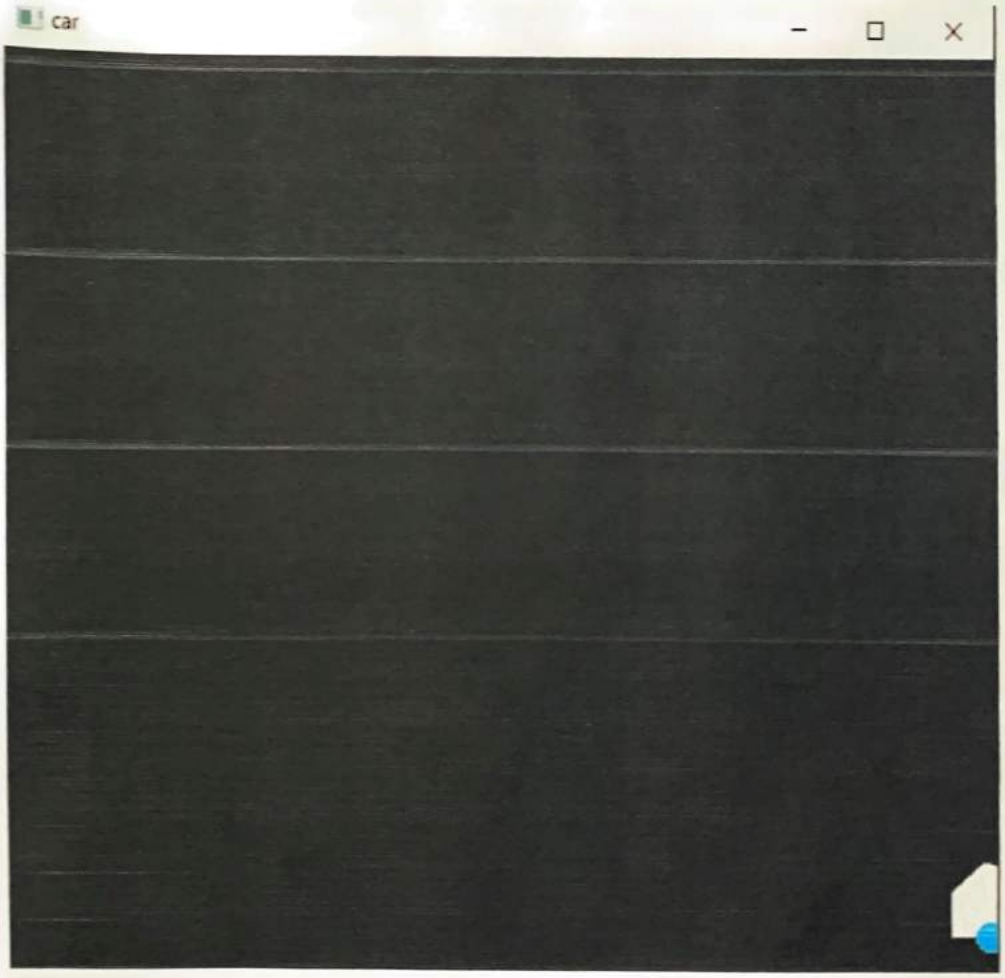
```
#include<GL/glut.h>
#include<math.h>
#include<stdio.h>
#defineCAR 1
#defineWHEEL 2
float s = 1;
voidcarlist() {
        glNewList(CAR, GL_COMPILE);
        glColor3f(1, 1, 1);
        glBegin(GL_POLYGON);
        glVertex3f(0, 25, 0);
        glVertex3f(90, 25, 0);
        glVertex3f(90, 55, 0);
        glVertex3f(80, 55, 0);
        glVertex3f(20, 75, 0);
        glVertex3f(0, 55, 0);
        glEnd();
        glEndList();
}
voidwheellist() {
        glNewList(WHEEL, GL_COMPILE_AND_EXECUTE);
        glColor3f(0, 1, 1);
        glutSolidSphere(10, 25, 25);
        glEndList();
}
voidmykeyboard(unsignedcharkey, intx, inty) {
        switch (key) {
        case't': glutPostRedisplay();
                break;
        case'q': exit(0);
        default: break;
        }
}
voidmyInit() {
        glClearColor(0, 0, 0, 0);
        glOrtho(0, 600, 0, 600, 0, 600);
}
voiddraw_wheel() {
        glColor3f(0, 1, 1);
        glutSolidSphere(10, 25, 25);
}
voidmoveCar(floats) {
        glTranslatef(s, 0.0, 0.0);
        glCallList(CAR);
        glPushMatrix();
        glTranslatef(25, 25, 0.0);      //move to first wheel position
        glCallList(WHEEL);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(75, 25, 0.0);     //move to 2nd wheel position
        glCallList(WHEEL);
        glPopMatrix();
        glFlush();
}
voidmyDisp() {
        glClear(GL_COLOR_BUFFER_BIT);
        carlist();
        moveCar(s);
        wheellist();
}
voidmouse(intbtn, intstate, intx, inty) {
        if (btn == GLUT_LEFT_BUTTON &&state == GLUT_DOWN) {
                s += 5;
                myDisp();
        }
        elseif (btn == GLUT_RIGHT_BUTTON &&state == GLUT_DOWN) {
                s += 2;
                myDisp();
        }
}
intmain(intargc, 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




car


car

PROGRAM - 10

a. Write a program to create a color cube and spin it using open GL 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,-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 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,
1.0,1.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,
1.0,0.0,1.0,1.0,1.0,1.0,0.0,1.0,1.0};

GLubyte cubeIndices[] = {0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,
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;
```

Teacher's Signature _____

```
void delay (float secs)
{
    float end = clock() / CLOCKS_PER_SEC + secs;
    while ((clock() / CLOCKS_PER_SEC) < end);
}
void displaying Single (void)
{
    /* display callback, clear frame buffer and z buffer,
       rotate cube and draw, swap buffers */

    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,
                                    cubeIndices);
    glBegin (GL_LINES);
    glVertex3f (0.0, 0.0, 0.0);
    glVertex3f (1.0, 1.0, 1.0);
    glEnd();
    glFlush();
}
void spinCube()
{
    /* Idle callback, spin cube 2 degrees about selected axis */
    // sleep (50);
    delay (0.01);
```

Teacher's Signature _____

```
    theta[axis] += 2.0;
    if (theta[axis] > 360.0) theta[axis] -= 360.0;
    glutPostRedisplay();
}

void mouse(int btn, int state, int x, int y)
{

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)
{
    // window 1
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("colorcube");
```

Teacher's Signature _____

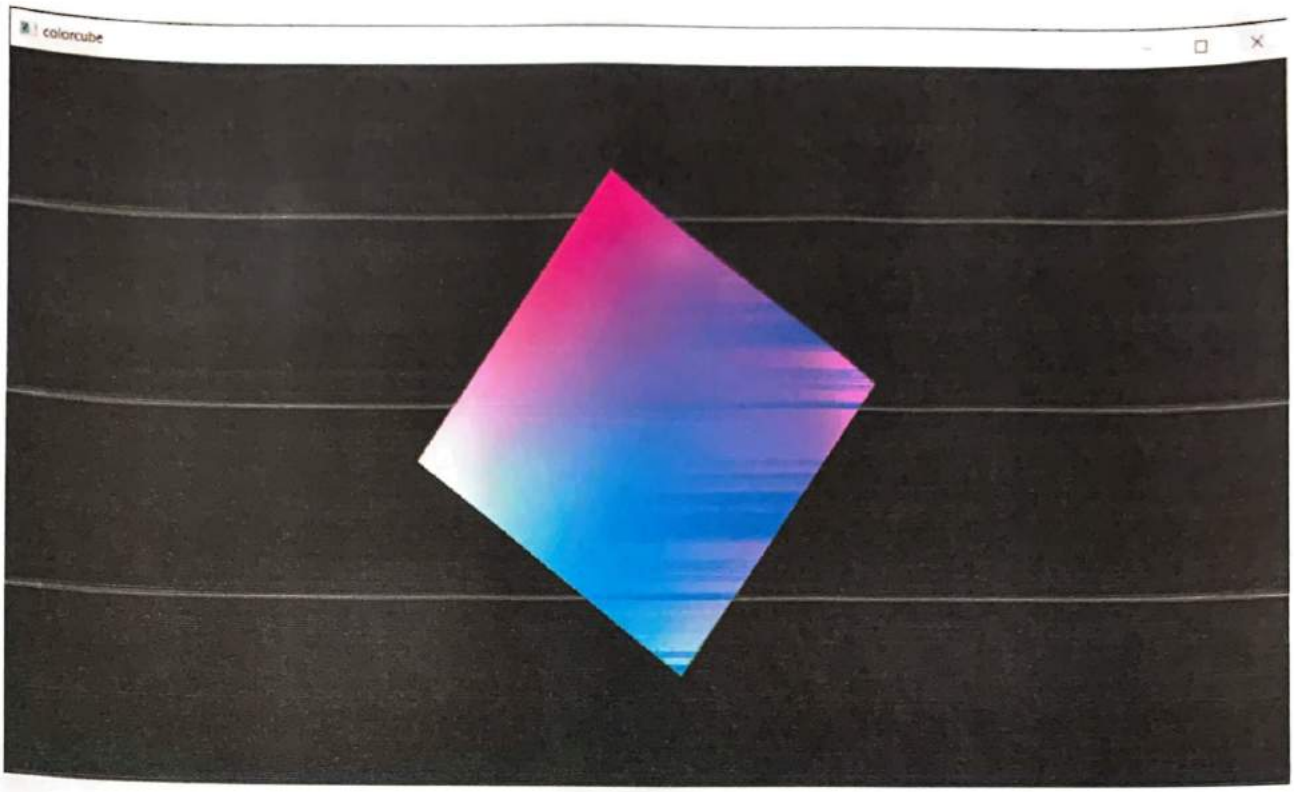
```
glutReshapeFunc (myReshape);
glutDisplayFunc (displaySingle);
glutIdleFunc (spinCube);
glutMouseFunc (mouse);
glEnable (GL_DEPTH_TEST); /* Enable hidden -- Surface
                             -- removal */
glEnableClientState (GL_COLOR_ARRAY);
glEnableClientState (GL_NORMAL_ARRAY);
glEnableClientState (GL_VERTEX_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 ();
}
```

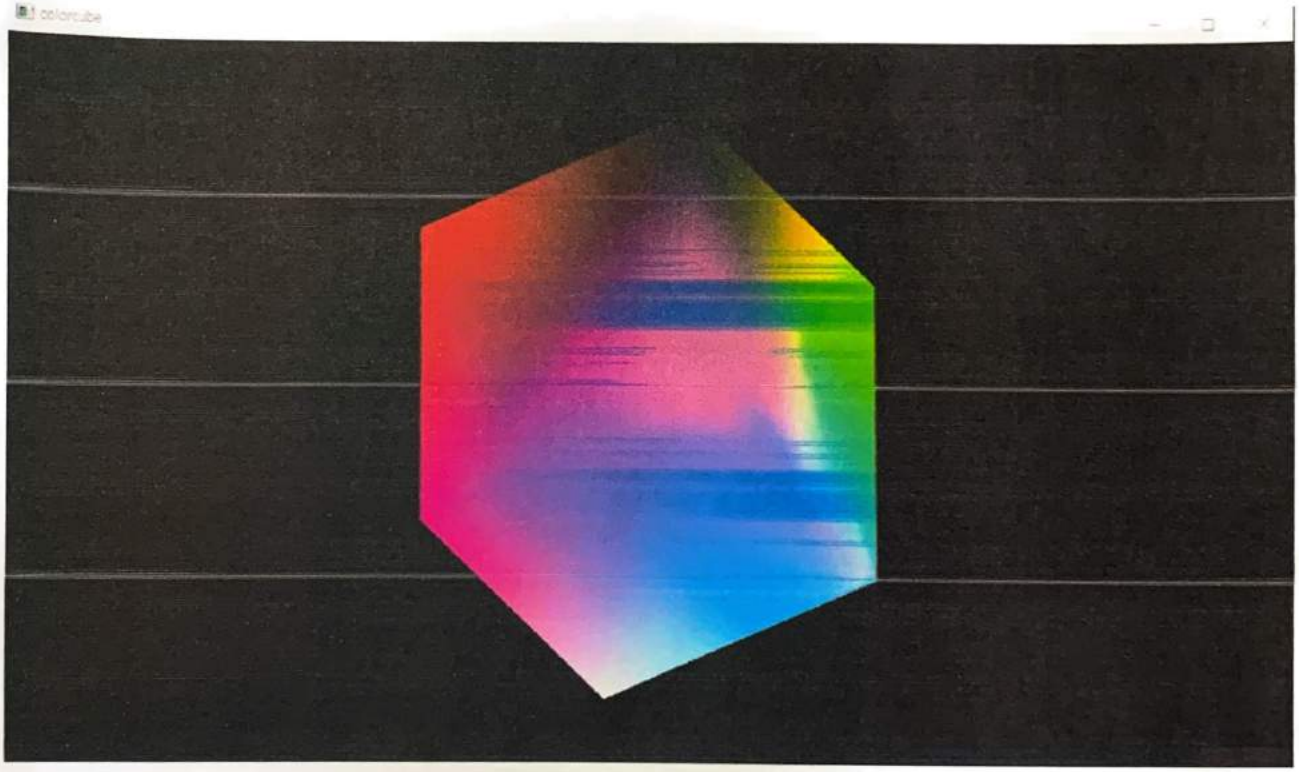
program 10

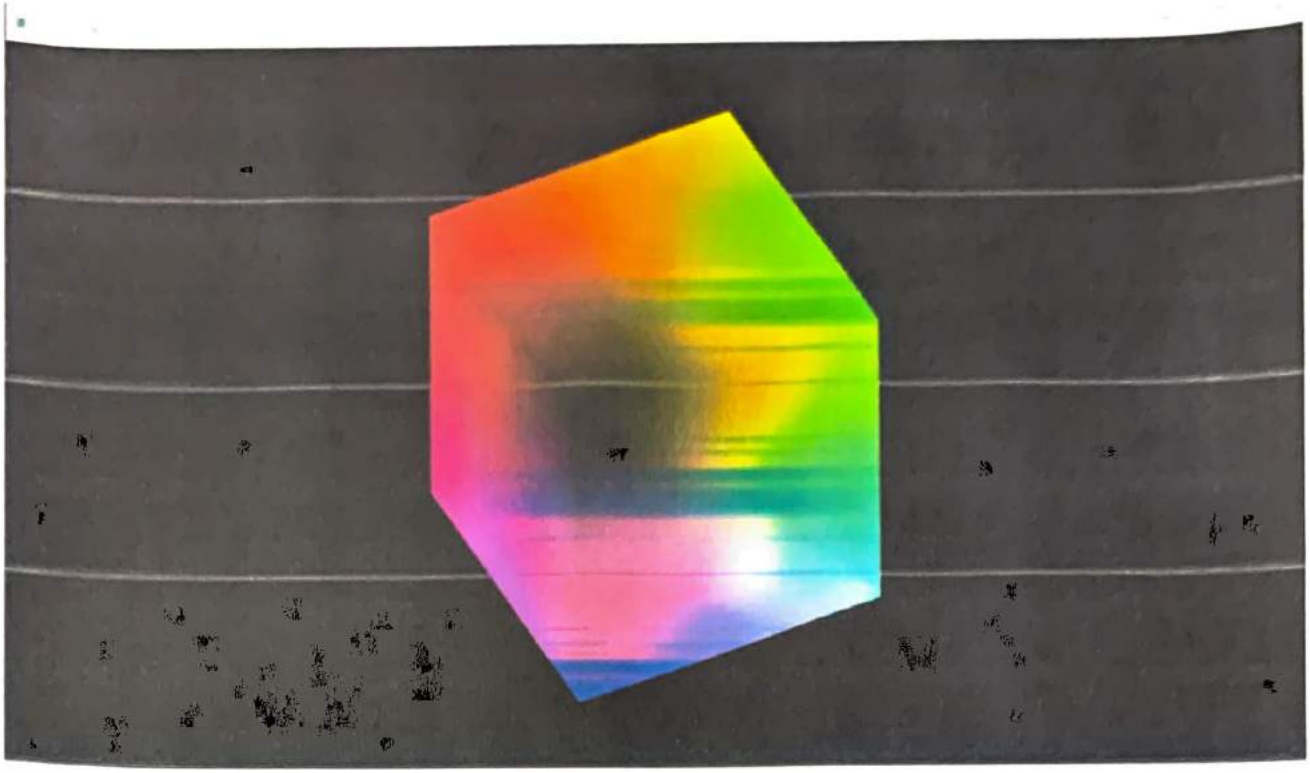
```
#include<stdlib.h>
#include<GL/glut.h>
#include<gl\GL.h>
#include<gl\GLU.h>
#include<time.h>
GLfloatvertices[] = { -1.0,-1.0,-1.0,1.0,-1.0,-1.0,
1.0,1.0,-1.0, -1.0,1.0,-1.0, -1.0,-1.0,1.0,
1.0,-1.0,1.0, 1.0,1.0,1.0, -1.0,1.0,1.0 };
GLfloatnormals[] = { -1.0,-1.0,-1.0,1.0,-1.0,-1.0,
1.0,1.0,-1.0, -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 };
GLfloatcolors[] = { 0.0,0.0,0.0,1.0,0.0,0.0,
1.0,1.0,0.0, 0.0,1.0,0.0, 0.0,0.0,1.0,
1.0,0.0,1.0, 1.0,1.0,1.0, 0.0,1.0,1.0 };
GLubytecubeIndices[] = { 0,3,2,1,2,3,7,6,0,4,7,3,1,2,6,5,4,5,6,7,0,1,5,4 };
staticGLfloattheta[] = { 0.0,0.0,0.0 };
staticGLfloatbeta[] = { 0.0,0.0,0.0 };
staticGLint axis = 2;
voiddelay(floatsecs)
{
        float end = clock() / CLOCKS_PER_SEC + secs;
        while ((clock() / CLOCKS_PER_SEC) < end);
}
voiddisplaySingle(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, cubeIndices);
        glBegin(GL_LINES);
        glVertex3f(0.0, 0.0, 0.0);
        glVertex3f(1.0, 1.0, 1.0);
        glEnd();glFlush();
}
voidspinCube()
{
        delay(0.01);
        theta[axis] += 2.0;
        if (theta[axis] > 360.0) theta[axis] -= 360.0;
        glutPostRedisplay();
}
voidmouse(intbtn, intstate, intx, inty)
{
        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;
}
voidmyReshape(intw, inth)
{
        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);
}
Voidmain(intargc, char** argv)
{
        //window 1
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowPosition(100, 100);
        glutInitWindowSize(500, 500);
        glutCreateWindow("colorcube");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(displaySingle);
        glutIdleFunc(spinCube);
        glutMouseFunc(mouse);
        glEnable(GL_DEPTH_TEST); /* Enable hidden--surface--removal */
        glEnableClientState(GL_COLOR_ARRAY);
        glEnableClientState(GL_NORMAL_ARRAY);
        glEnableClientState(GL_VERTEX_ARRAY);
        glVertexPointer(3, GL_FLOAT, 0, vertices);
        glColorPointer(3, GL_FLOAT, 0, colors);
```

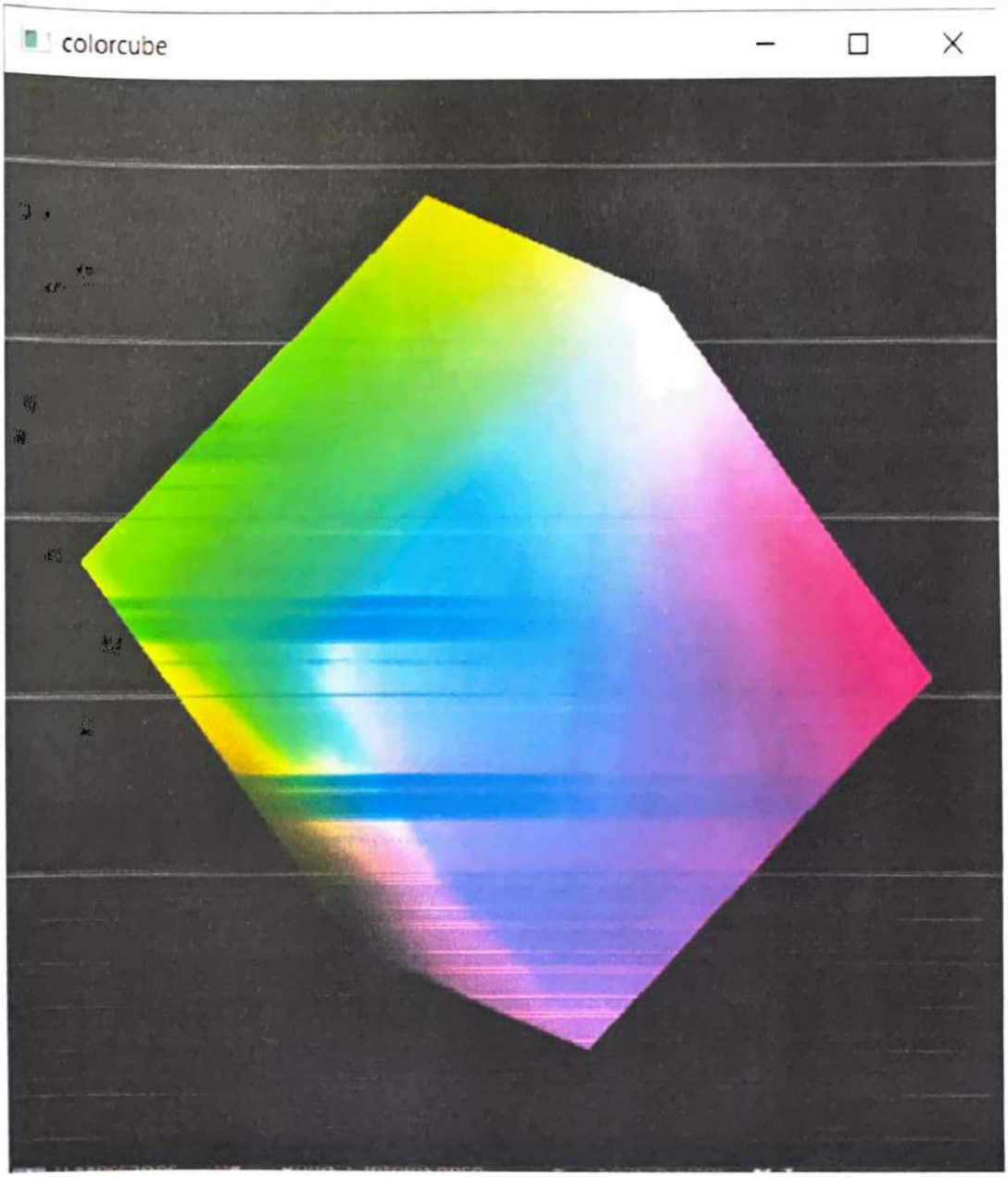
```
        glNormalPointer(GL_FLOAT, 0, normals);
        glColor3f(1.0, 1.0, 1.0);
}       glutMainLoop();
```

Output








colorcube

PROGRAM - 11

a Create a menu with three entries named curves, colors and quit. The Entry Curves has a Sub menu which has four Entries namely Limacon, Three-Leaf, Cardiod 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 the mouse buttons.

```
#include <gl/glut.h>
#include <math.h>
#include <stdio.h>
struct Screenpt {
        int x;
        int y;
};
typedef enum { limacon=1, Cardioid=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);
        gl vertex2i(p1.x, p1.y);
        gl vertex2i(p2.x, p2.y);
```

Teacher's Signature

```
    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;
    curve pt[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;
    }
```

Teacher's Signature _____

```
        curvept[i].x = x0 + r* cos(theta);
        curvept[i].y = y0 + r* sin(theta);
        lineSegment(curvept[0], curvept[1];

        curvept[0].x = x0 + r* cos(theta);
        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 4:
        red = 1;
        green = 0;
        blue = 0;
        break;
```

```
case 3:
        red = 0;
        green = 1;
        blue = 1;
        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;
}
draw Curve(curve);
}
void main_menu(int id) {
        switch(id) {
        case 3; exit(0);
        default: break;
        }
```

```
}
void mydisplay() {
    /* int curvenum = 1;
    glClear(GL_COLOR_BUFFER_BIT);
    /* printf("Enter the integer value corresponding to one
       of the following curve names:\n");
    Printf("1 - limacon\n2 - cardioid\n3 - threeleaf\n4 -
                                        Spiral\n");
    Scanf_s("%d", &curveNum);*/
    /* if (curveNum == 1 || CurveNum == 2 || curveNum == 3 ||
       CurveNum == 4) drawCurve(curvenum);*/
}
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 = glutCreatMenu(drawCurve);
    glutAddMenuEntry("Limacon", 1);
    glutAddMenuEntry("Cardioid", 2);
    glutAddMenuEntry("Threeleaf", 3);
    glutAddMenuEntry("Spiral", 4);
```

Teacher's Signature _____

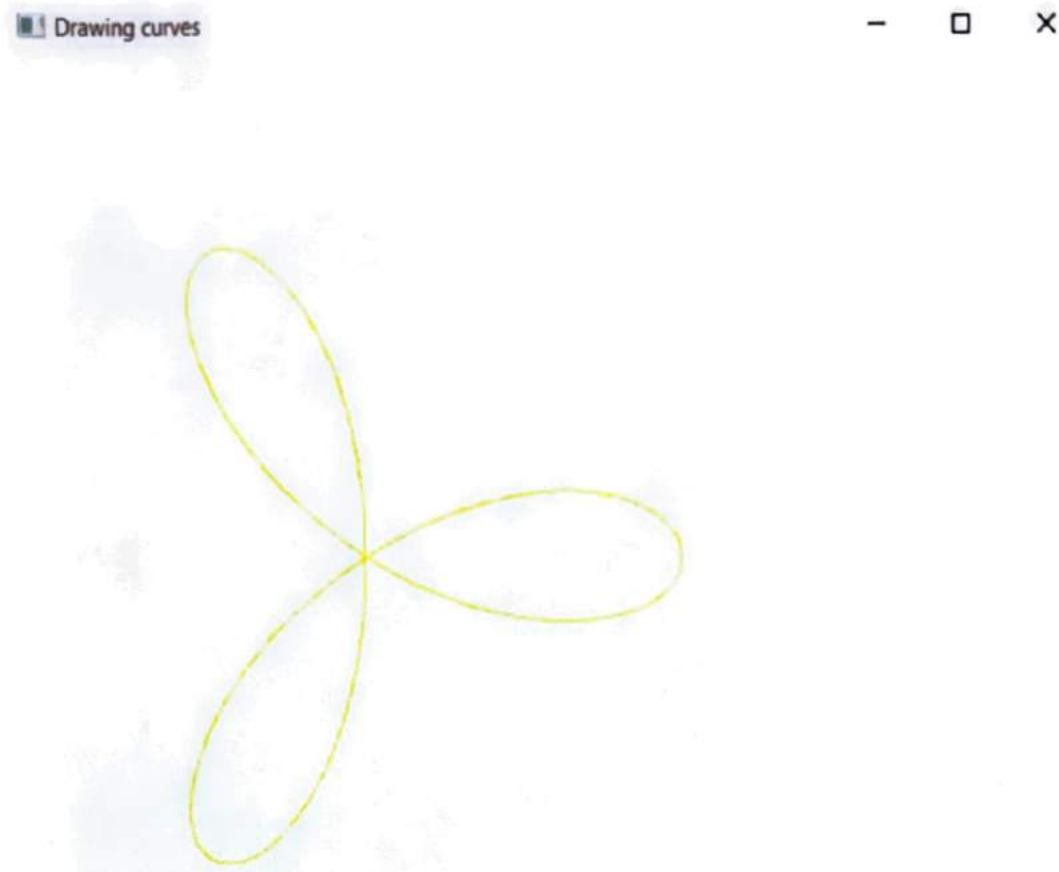
```
glutAttachMenu(GLUT_LEFT_BUTTON);
int colorId = glutCreateMenu(colorMenu);
glutAddMenuEntry("Red",4);
glutAddMenuEntry("Green",2);
glutAddMenuEntry("Blue",1);
glutAddMenuEntry("Black",0);
glutAddMenuEntry("yellow",6);
glutAddMenuEntry("cyan",3);
glutAddMenuEntry("Magenta",5);
glutAddMenuEntry("white",7);
glutAddMenuEntry(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();

}
```

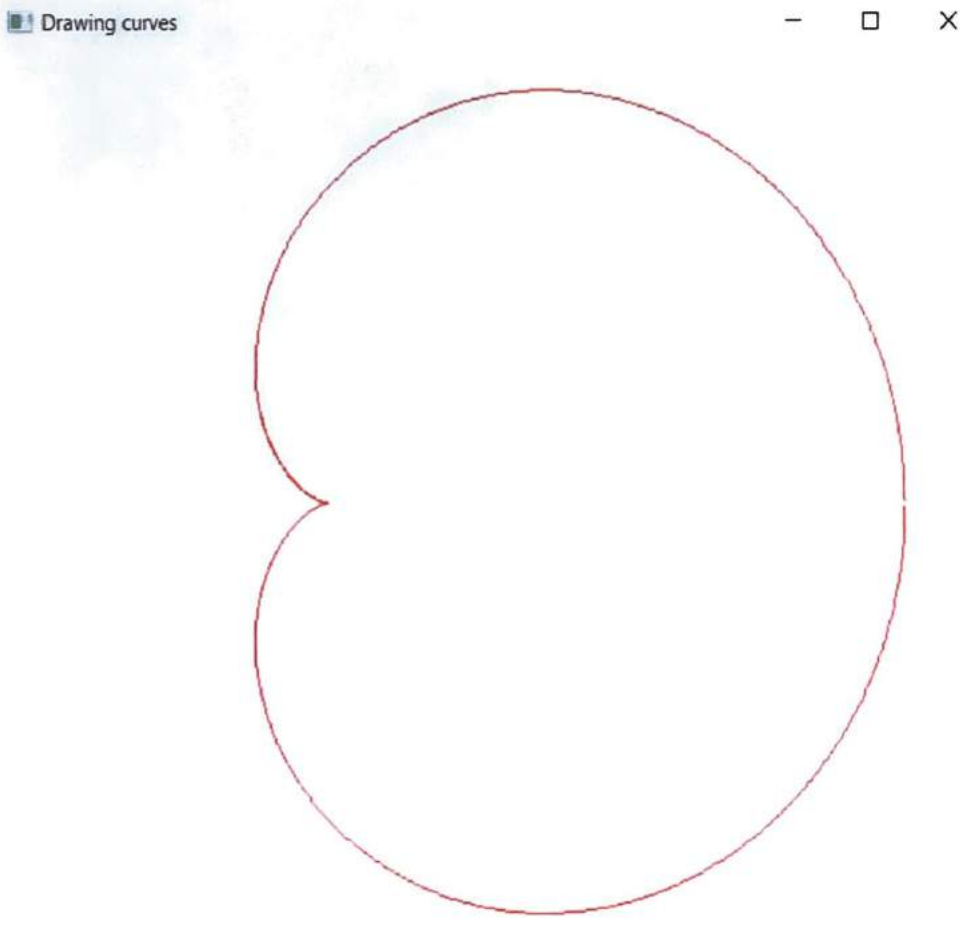
program 11

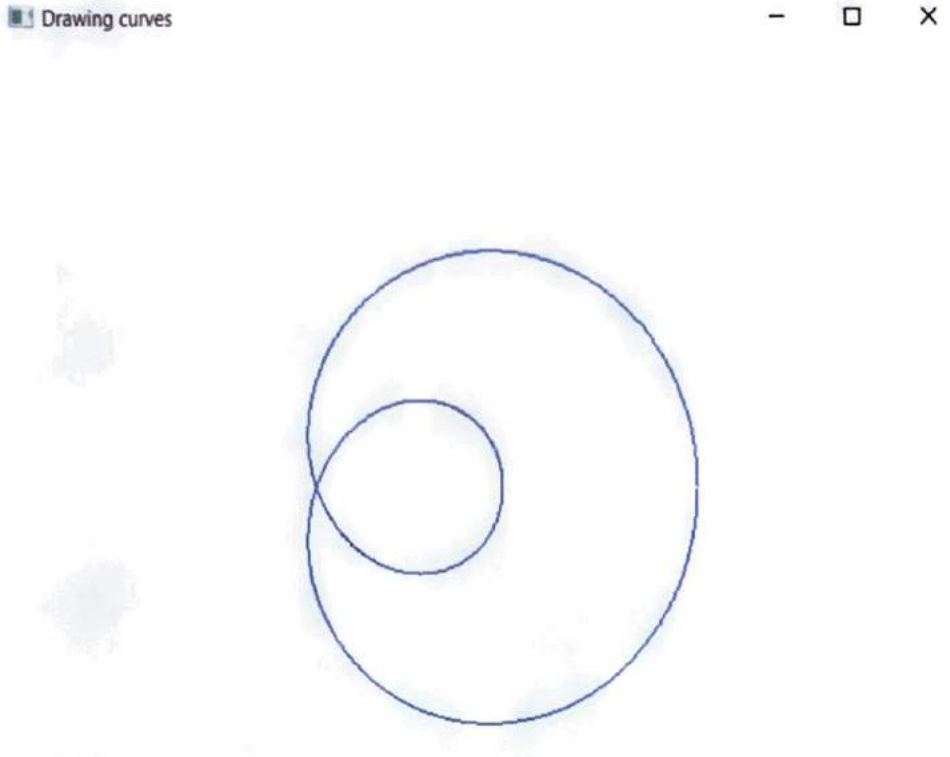
```
#include<gl/glut.h>
#include<math.h>
#include<stdio.h>
structscreenPt {
        int x;
        int y;
};
typedefenum{ limacon = 1, cardioid = 2, threeLeaf = 3, spiral = 4 } curveName;
int w = 600, h = 500;
int curve = 1;
int red = 0, green = 0, blue = 0;
voidmyinit(void) {
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}
voidlineSegment(screenPtp1, screenPtp2) {

        glBegin(GL_LINES);
        glVertex2i(p1.x, p1.y);
        glVertex2i(p2.x, p2.y);
        glEnd();
        glFlush();
}
voiddrawCurve(intcurveNum) {
        constdoubletwoPi = 6.283185;
        constint a = 175, b = 60;
        float r, theta, dtheta = 1.0 / float(a);
        int x0 = 200, y0 = 250;
        screenPtcurvePt[2];
        curve = curveNum;
        glColor3f(red, green, blue);
        curvePt[0].x = x0;
        curvePt[0].y = y0;
        glClear(GL_COLOR_BUFFER_BIT);
        switch (curveNum) {
        caselimacon: curvePt[0].x += a + b; break;
        casecardioid: curvePt[0].x += a + a; break;
        casethreeLeaf: curvePt[0].x += a; break;
        casespiral: break;
        default: break;
        }
        theta = dtheta;
        while (theta <twoPi) {
                switch (curveNum) {
                caselimacon: r = a * cos(theta) + b; break;
                casecardioid: r = a * (1 + cos(theta)); break;
                casethreeLeaf: r = a * cos(3 * theta); break;
                casespiral: 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;
        }
}

voidcolorMenu(intid) {
        switch (id) {

        case 0:
                break;
        case 1:
                red = 0;
                green = 0;
                blue = 1;

                break;
        case 2:
                red = 0;
                green = 1;
                blue = 0;
                break;

        case 4:
```

```
                red = 1;
                green = 0;
                blue = 0;

                break;
        case 3:
                red = 0;
                green = 1;
                blue = 1;

                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);
}
voidmain_menu(intid) {

        switch (id) {

        case 3: exit(0);
        default: break;
        }
}
voidmydisplay() {}

voidmyreshape(intnw, intnh) {
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, (double)nw, 0.0, (double)nh);
        glClear(GL_COLOR_BUFFER_BIT);
}
voidmain(intargc, char** argv) {
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(w, h);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("Drawing curves");
        intcurveId = glutCreateMenu(drawCurve);
        glutAddMenuEntry("Limacon", 1);
        glutAddMenuEntry("Cardioid", 2);
        glutAddMenuEntry("Threeleaf", 3);
        glutAddMenuEntry("Spiral", 4);
        glutAttachMenu(GLUT_LEFT_BUTTON);
        intcolorId = glutCreateMenu(colorMenu);
        glutAddMenuEntry("Red", 4);
        glutAddMenuEntry("Green", 2);
        glutAddMenuEntry("Blue", 1);
        glutAddMenuEntry("Black", 0);
        glutAddMenuEntry("Yellow", 6);
        glutAddMenuEntry("Cyan", 3);
        glutAddMenuEntry("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






Drawing curves


Drawing curves

Expt. No. _____ Date _____

Page No. 55

PROGRAM - 12

Q Write a Program to construct Bezier Curve. Control points are supplied through keyboard mouse.

```
#include<iostream>
#include<math.h>
#include<gl/glut.h>

using namespace std;
float f,g,r,xc[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;
```

Teacher's Signature _____

```
double t;
gl Color3f(0.0,0);
gl Begin(GL_POINTS);
for (t=0; t<1; t=t+0.005) {
        double xt = pow(1-t,3)*x1[0]+3*t*pow
3*pow(t,2)*(1-t)*x1[2]+pow(t,3)*x1(3];1-t,2)*x1[1]+
        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];
        gl Vertex2f(xt, yt);
    }
    gl Color3f(1,1,0);
    for (i=0; i<4; i++) {
     gl End();
     gl Flush();
    }
}
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;
        gl PointSize(3);
        gl Color3f(1,1,0);
        gl Begin(GL_POINTS);
        gl Vertex2i(x, 500-y);
        gl End();
        gl Flush();
        flag++;
```

Teacher's Signature _____

```
    }
    if(flag >=4 && btn == GLUT_LEFT_BUTTON)
    {
        glColor3f(0.0,0.1);
        display();
        flag = 0;
    }
}
Int Main(int argc; char* argv[]) {
    glutInit(& argc, argv);

    /*
    // USE KEYBOARD
    cout << "Enter the x co-ordinates";
    cin >> x1[0] >> x1[1] >> x1[2] >> x1[3];
    cout << "Enter y co-ordinates";
    cin >> yc[0] >> yc[1] >> yc[2] >> yc[3]
    // END KEYBOARD
    */
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("B2");
    glutDisplayFunc(display);
    glutMouseFunc(mymouse);// INCLUDE FOR MOUSE,
                           REMOVE FOR KEYBOARD
    MyInit();
    glutMainLoop();
}
```

program 12

```
#include<iostream>
#include<math.h>
#include<gl/glut.h>
usingnamespace std;
float f, g, r, x1[4], yc[4];
int flag = 0;
voidmyInit() {

	glClearColor(1, 1, 1, 1);
	glColor3f(1, 1, 1);
	glPointSize(5);
	gluOrtho2D(0, 500, 0, 500);
}

voiddrawPixel(floatx, floaty) {
	glBegin(GL_POINTS);
	glVertex2f(x, y);
	glEnd();
}

voiddisplay() {
	glClear(GL_COLOR_BUFFER_BIT);
	inti;
	double t;
	glColor3f(0, 0, 0);
	glBegin(GL_POINTS);
	for (t = 0; t < 1; t = t + 0.005) {
		doublext = 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];
		doubleyt = 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();
	}
}
voidmymouse(intbtn, intstate, intx, inty)
{
	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, 1, 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;
	}
}
intmain(intargc, char* argv[]) {
	glutInit(&argc, argv);
	//USE KEYBOARD
	cout<<"Enter the x co-ordinates";
	cin >> x1[0] >> x1[1] >> x1[2] >> x1[3];
	cout<<"Enter y co-ordinates";
	cin>>yc[0] >>yc[1] >>yc[2] >>yc[3];
	//END KEYBOARD
	glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
	glutInitWindowSize(500, 500);
	glutInitWindowPosition(0, 0);
	glutCreateWindow("BZ");
	glutDisplayFunc(display);
	//glutMouseFunc(mymouse); //INCLUDE FOR MOUSE, REMOVE FOR KEYBOARD
	myInit();
	glutMainLoop();
}
```

output