PGM 1 INPUT: Enter points: 50 50 100 100 40 60 90 30
92 35 110 100

OUTPUT

EXPT. NO. 1 NAME: BRESENHAM'S LINE DRAWING Page No. Date: 23-10-20

PROGRAM - 1

Write a program to generate a line using Bresenham's line drawing technique. Consider slopes greater than one. User must be able to draw as many lines and specify inputs through keyboard/mouse.

CODE:

```
#include <iostream.h>
#include <GLUT/GLUT.h>
#include <time.h>
using namespace std;
int x1, x2, y1, y2;
int flag = 0;

void draw_pixel (int x, int y)          //to draw each point
{   glColor3f (1, 0, 0);
    glBegin (GL_POINTS)
    glVertex2i (x, y);
    glEnd ();
    glFlush ();
}

void draw_line ()                       //uses Bresenham's algo to plot points
{   int dx, dy, i, ~~b~~;
    int incx, incy, inc1, inc2;
```

Teacher's Signature

```
#include <iostream>
#include <GLUT/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;
        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;
```

EXPT. NO. NAME: Page No. Date YOUVA

```
int x,y;
  dx = x2-x1;
  dy = y2-y1;
  if(dx<0) dx = -dx;
  if(dy<0) dy = -dy;
  incx = 1;
  if(x2<x1) incx = -1;
  incy = 1;
  if(y2<y1) incy = -1

  x = x1;
  y = y1;
  if(dx>dy)                    // slope m < 1
    {   ~~draw_pixel(x,y);~~
        p = 2*dy - dx;         // setting decision parameter
        inc1 = 2*(dy-dx);
        inc2 = 2*dy;
        for(i=0; i<dx; i++)
          {  draw_pixel(x,y);
             if(p>0)
               { y+= incy;
                 p+= inc1;
               }
             else
                 p+= inc2;
             x+= incx;
          }
    }
```

Teacher's Signature

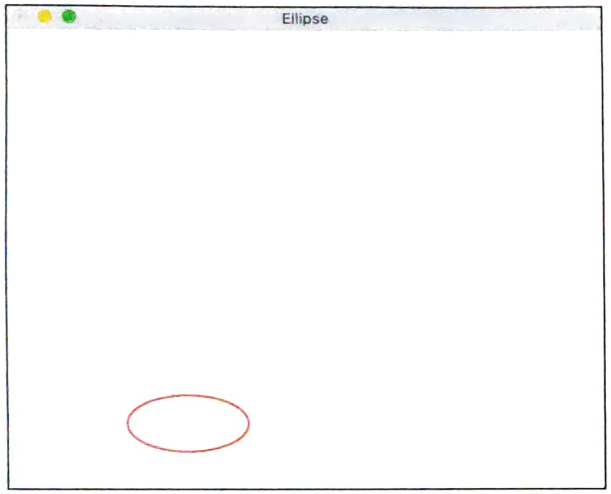
EXPT. NO. NAME: Page No: Date: YOUVA

```
else {  p = 2*dx - 2y;                    //slope greater than 1
        inc1 = 2*(dx-dy);
        inc2 = 2*dy;
    for (i=0; i<dy; i++){
          draw_pixel(x,y);
           if (p>0)
           { x+=incx; p+=inc1;}
      else   p+=inc2;
          y+=incy;
                  }
    }
    glFlush(); }

Void myInit(){ glClear(GL_COLOR_BUFFER_BIT), //clear buffer
  glClearColor(1,1,1,1); //White BG
  gluOrtho2D(0,500,0,500);
}
int main(int argc, char* argv[])
{  printf("enter points: x1 y1 x2 y2 \n");
   scanf("%d %d %d %d", &x1, &y1, &x2, &y2);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,200);
    glutCreateWindow("Bresenham's Algo");
     myinit();
    glutInit glutKeyboardFunc(key) glutDisplayFunc(draw_line);
     glutMainLoop();
}
```

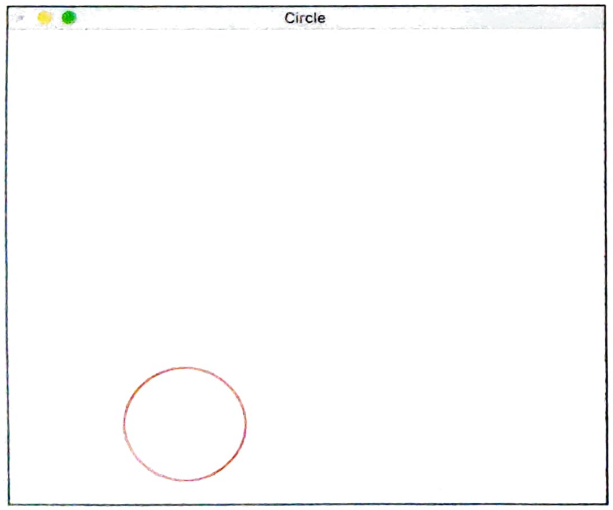
Teacher's Signature:

INPUT: Enter 1 to draw a circle, 2 to draw ellipse 2 Enter center & major & minor radius
OUTPUT

100 50 50 25



Enter 1 to draw circle, 2 to draw ellipse 1 Enter center & radius
100 50 50



EXPT. NO. 2 NAME: BRESENHAM'S CIRCLE & ELLIPSE DRAWING TECHNIQUE

Page No.
Date

YOUVA

PROGRAM - 2

Write a program to generate a circle and ellipse using Bresenheim'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 <GLUT/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) //plotting points decided by algo
{
    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();
}
```

Teacher's Signature

```c
#include<GLUT/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);
    glVertex2i(xc + x, yc + y);
    glVertex2i(xc - x, yc + y);
    glVertex2i(xc + x, yc - y);
    glVertex2i(xc - x, yc - y);
    glVertex2i(xc + y, yc + x);
    glVertex2i(xc - y, yc + x);
    glVertex2i(xc + y, yc - x);
    glVertex2i(xc - y, yc - x);
    glEnd();
}
void circlebres()
{
    glClear(GL_COLOR_BUFFER_BIT);
    int x = 0, y = r;
    int d = 3 - 2 * r;
    while (x <= y)
    {
        draw_circle(xc, yc, x, y);
        x++;
        if (d < 0)
          d = d + 4 * x + 6;
        else
        {
            y--;
            d = d + 4 * (x - y) + 10;
        }
        draw_circle(xc, yc, x, y);
    }
    glFlush();
}
int p1_x, p2_x, p1_y, p2_y;
int point1_done = 0;
void myMouseFunccircle(int button, int state, int x, int y)
{
    if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && point1_done == 0)
    {
        p1_x = x - 250;
        p1_y = 250 - y;
        point1_done = 1;
    }
    else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
    {
        p2_x = x - 250;
        p2_y = 250 - y;
        xc = p1_x;
        yc = p1_y;
        float exp = (p2_x - p1_x) * (p2_x - p1_x) + (p2_y - p1_y) * (p2_y - p1_y);
        r = (int)(sqrt(exp));
        circlebres();
        point1_done = 0;
    }
}


void draw_ellipse(int xce, int yce, int x, int y)
{
```

EXPT. NO | NAME: | Page No | Date | YOUVA

```
void circlebres()  //Bresenham algo for plotting circle
{
    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();
}


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

Teacher's Signature:

EXPT. NO. NAME: Page No. Date. YOUVA

```
void midpt ellipse()        //algo for plotting ellipse
{ glClear(GL_COLOR_BUFFER_BIT);
  float dx, dy, d1, d2, x, y;
   x=0;
   y=ry;
  // decision parameter of region 1
   p1 = (ry*ry) - (rx*rx*ry) + (0.25*rx*rx);
   dx = 2*ry*ry*x          //2ry^2x
   dy = 2*rx*rx*y          //2rx^2y

   while (dx < dy)         //plotting points of region 1
    { draw_ellipse(xc, yc, x, y);
      if (p1 < 0)          //checking & updating parameter
     { x++;                //based on algo. pt inside
      dx = dx + (2*ry*ry);
      p1 = p1 + dx + (ry*ry);
     }
      else                 // pt outside side, take y-1
     { x++;
       y--;
       dx = dx + (2*ry*ry);
       dy = dy - (2*rx*rx);
       p1 = p1 + dx - dy + (ry*ry);
     }
    }
  // decision parameter of region 2
   p2 = ((ry*ry)*((x+0.5)*(x+0.5))) + ((rx*rx)*((y-1)*(y-1))) -
                                  (rx*rx*ry*ry);
```

Teacher's Signature

```
while (y >= 0)        // plotting points of regions
{   draw_ellipse (xce, yce, x, y);      // plot points 4-way symmetry

    if (p2 > 0)       // checking & updating parameter based on algo
    {   y--;                            // pt outside
        dy = dy - (2 * ry * ry);
        p2 = p2 + (rx * rx) - dy;
    }
    else
    {   y--;                            // pt inside take x_k + 1
        x++;
        dx = dx + (2 * ry * ry);
        dy = dy - (2 * rx * rx);
        p2 = p2 + dx - dy + (rx * rx);
    }
}
    glFlush();
}

void myinit()
  { glClearColor (1, 1, 1, 1);          // White Bg
    glColor3f (1.0, 0.0, 0.0);          // Red color
    glPointSize (3.0);
    gluOrtho2D (0, 400, 0, 400);
  }

int main (int argc, char* argv[])
  {   glutInit(&argc, argv);
```

Teacher's Signature:

EXPT. NO. | NAME: | Page No. | Date: | YOUVA

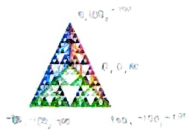
```
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (0, 0);

printf(" enter 1 to draw a circle, 2 to draw ellipse\n");
int ch;
 scanf ("%d", &ch);
 switch(ch)
 {
   case 1: printf (" Enter centre coordinates and radius\n");
           scanf (" %d %d %d", &xc, &yc, &r);
           glutCreateWindow (" Circle");
           glutDisplayFunc ( circlebres);
           break;
   case 2: printf (" Enter centre coordinates & major and minor radius\n");
           scanf (" %d %d %d %d", &xce, &yce, &rx, &ry);
            glutCreateWindow (" Ellipse");
            glutDisplayFunc (midptellipse);
            break;
  }
  myinit ();
  glutMainLoop ();
}
```

Teacher's Signature

PGM 3

INPUT: Enter number of iterations : 6

OUTPUT



EXPT. NO. 3 NAME: SEIRPINSKI 3D GASKET Page No. Date YOUVA

PROGRAM - 3

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

```
#include <GLUT/GLUT.h>
#include <stdio.h>
int n;
typedef float point[3];
point vtx[4] = {{0,100,-100},{0,0,100},{100,-100,-100},{-100,-100,-100}};

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

void divide_triangle(point a, point b, point c, int m)
{   point v1, v2, v3;
    int j;
    if(m>0)
    {   for(j=0;j<3;j++)
          v1[j]=(a[j]+b[j])/2;      //midpt of each side
        for(j=0;j<3;j++)
          v2[j]=(a[j]+c[j])/2;
```

Teacher's Signature:

```c
#include<gl/glut.h>
#include<GLUT/GLUT.h>
#include<stdio.h>

int m;
typedef float point[3];
point tetra[4] = { {0,100,-100},{0,0,100},{100,-100,-100},{-100,-100,-100} };
void tetrahedron(void);
void myinit(void);
void divide_triangle(point a, point b, point c, int m);
void draw_triangle(point p1, point p2, point p3);
int main(int argv, char** argc)
{
    //int m;
    printf("Enter the number of iterations: ");
    scanf("%d", &m);
    glutInit(&argv, argc);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(100, 200);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Seirpinski Gasket");
    glutDisplayFunc(tetrahedron);
    glEnable(GL_DEPTH_TEST);
    myinit();
    glutMainLoop();
}
void divide_triangle(point a, point b, point c, int m)
{
    point v1, v2, v3;
    int j;
    if (m > 0) {
        for (j = 0; j < 3; j++)
            v1[j] = (a[j] + b[j]) / 2;
        for (j = 0; j < 3; j++)
            v2[j] = (a[j] + c[j]) / 2;
        for (j = 0; j < 3; j++)
            v3[j] = (b[j] + c[j]) / 2;

        divide_triangle(a, v1, v2, m - 1);
        divide_triangle(c, v2, v3, m - 1);
        divide_triangle(b, v3, v1, m - 1);
    }
    else
        draw_triangle(a, b, c);
}
void myinit()
{
    glClearColor(1, 1, 1, 1);

    glOrtho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);
}
void tetrahedron(void)
{
    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_triangle(tetra[3], tetra[2], tetra[1], m);
    glColor3f(0.0, 0.0, 1.0);
    divide_triangle(tetra[0], tetra[3], tetra[1], m);
    glColor3f(0.0, 0.0, 0.0);
    divide_triangle(tetra[0], tetra[2], tetra[3], m);
    glFlush();
}
```

EXPT. NO. | NAME: | Page No. | Date: | YOUVA

```
for(j=0; j<3; j++)
    v3[j] = (b[j]+c[j])/2;

divide_triangle(a, v1, v2, m-1);   // recursively calls same
divide_triangle(c, v2, v3, m-1);   // function
divide_triangle(b, v3, v1, m-1);
}
else draw_triangle(a, b, c);
}

void tetrahedron(void)              // draw 3D seirpinski gasket
{ 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_triangle(tetra[3], tetra[2], tetra[1], m);
  glColor3f(0.0, 0.0, 1.0);
  divide_triangle(tetra[0], tetra[3], tetra[1], m);
  glColor3f(0.0, 0.0, 0.0);
  divide_triangle(tetra[0], tetra[2], tetra[3], m);
  glFlush();
}

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

Teacher's Signature:

EXPT. NO. | NAME: | Page No. | Date. | YOUVA

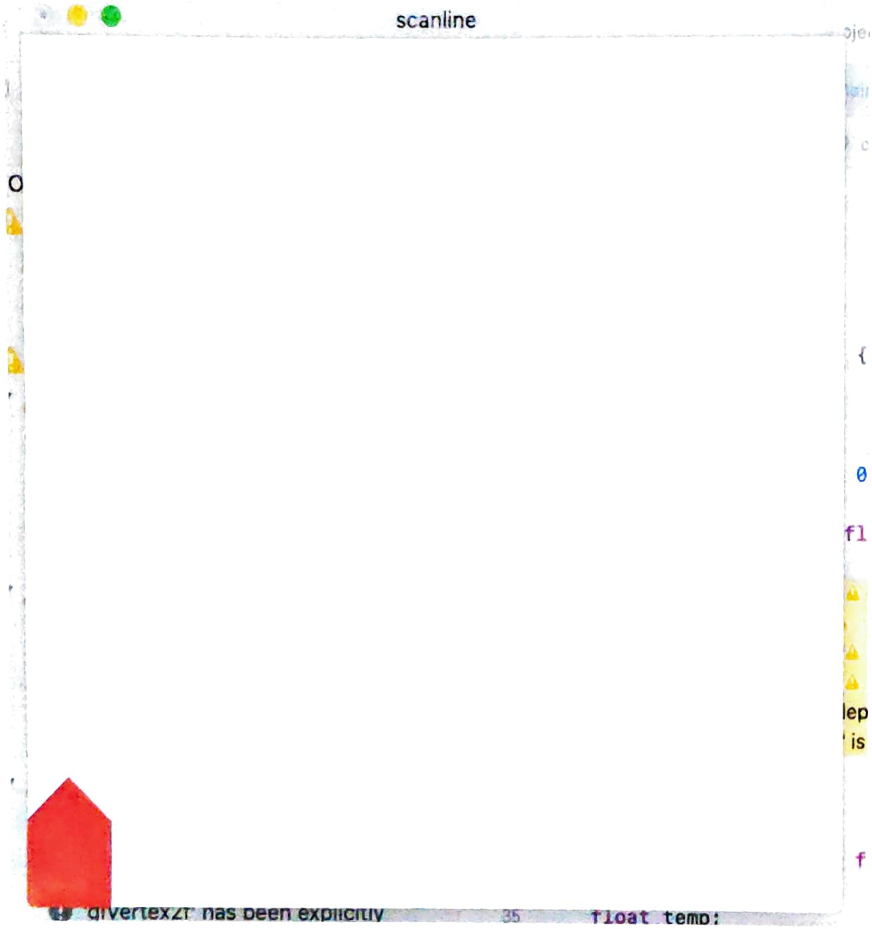
```
int main ( int argv, char* argc[])
{
    printf ("Enter number of iterations : ");
    scanf("%d", &n);
    glutInit (&argv, &argc);
    glutInitDisplayMode ( GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition (100, 200);
    glutInitWindowSize ( 500, 500 );
    glutCreateWindow ("SeinPinki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

Teacher's Signature

PGM 4

INPUT: Enter no of vertex : 5 Enter coordinates : {0, 0}{100, 0}{100, 100}{50, 150}{0, 100}

OUTPUT



EXPT NO 4 | NAME SCAN-LINE AREA FILLING ALGO | Page No. | Date | YOUVA

PROGRAM - 4

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

```
#include <GLUT/GLUT.h>
#include <stdlib.h>
#include <algorithm>
#include <iostream>

using namespace std;
float x[100], y[100];
int wx = 500, wy = 500;
int n, m;
static float intx[10] = {0};

void draw_line (float x1, float y1, float x2, float y2)
{   glColor3f (1, 0, 0);                    //draw line between
    glBegin (GL_LINES);                     //specified end pts
    glVertex2f (x1, y1);
    glVertex2f (x2, y2);
    glEnd ();
    glFlush ();
}

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

Teacher's Signature

```
#include<GLUT/GLUT.h>
#include<stdlib.h>
#include<algorithm>
#include<iostream>


using namespace std;
float x[100], y[100];

int n, m;
int wx = 500, wy = 500;
static float intx[10] = { 0 };

void draw_line(float x1, float y1, float x2, float y2) {

    glColor3f(1, 0, 0);
    glBegin(GL_LINES);
    glVertex2f(x1, y1);
    glVertex2f(x2, y2);
    glEnd();
    glFlush();

}

void edgeDetect(float x1, float y1, float x2, float y2, int scanline) {

    float temp;
    if (y2 < y1) {

        temp = x1; x1 = x2; x2 = temp;
        temp = y1; y1 = y2; y2 = temp;
    }

    if (scanline > y1 && scanline < y2)
        intx[m++] = x1 + (scanline - y1) * (x2 - x1) / (y2 - y1);
}

void scanfill(float x[], float y[]) {
    for (int s1 = 0; s1 <= wy; s1++) {
        m = 0;
        for (int i = 0; i < n; i++) {

            edgeDetect(x[i], y[i], x[(i + 1) % n], y[(i + 1) % n], s1);
        }
        sort(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() {

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

}
```

EXPT. NO. | NAME: | Page No. | Date: | YOUVA

```
    { 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); // get intersection point with scanline
}


void scanfill ( float x[], float y[])
{   for (int s1 = 0; s1 <= wy; s1++)
      { m = 0;
        for (int i = 0; i < n; i++)          //iterates through each edge
          { edgeDetect(x[i], y[i], x[(i+1)%n], y[(i+1)%n], s1);
          }
        sort( intx, (intx+m));               // sort intersection pts
        if(m >= 2)
          for(int i=0; i < m; i= i+2)
            draw_line (intx[i], s1, intx[i+1], s1); // draw line b/w
                                                    //each pair of pts
      }
}


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

Teacher's Signature:

EXPT. NO. | NAME: | Page No.: | Date: | VOUVA

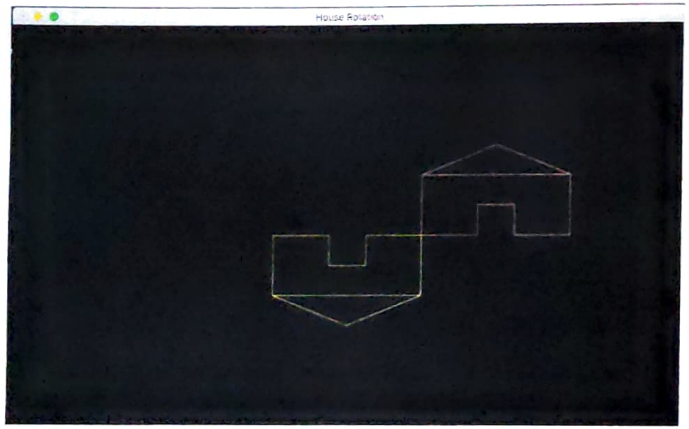
```
void myInit()
{
    glClearColor(1,1,1,1);
    glColor3f(0,0,1);
    glPointSize(1);
    gluOrtho2D(0, wx, 0, wy);
}

int main(int argc, char* argv[])
{
    glutInit(&argc, argv);
    printf("Enter no of edges:\n")
    scanf("%d", &n);
    printf("Enter coordinates of endpoints:\n");
    for(int i=0; i<n; i++)
    {   printf("X-coord Y-coord:\n");
        scanf("%f %f", &x[i], &y[i]);
    }
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Scanline");
    glutDisplayFunc(display_filled_polygon);
    myInit();
    glutMainLoop();
}
```

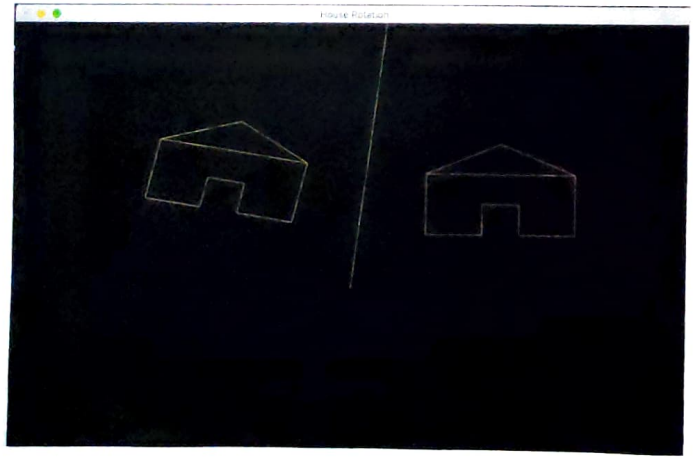
Teacher's Signature:

INPUT

```
(base) ruchitabiradar@Ruchitas-MacBook-Pro CG Lab % ./pgm5
Enter the rotation angle
180
Enter c and m value for line y=mx+c
8 10
^C
```

OUTPUT





EXPT. NO. 5 NAME: Page No. Date

PROGRAM - 5

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

```
#include <stdio.h>
#include <GLUT/glut.h>
#include <math.h>

float house[11][2] = { {100,200},{200,250},{300,200},{100,200},{100,100},
{175,100},{175,150},{225,150},{225,100},{300,100},{300,200}};
int angle=50;
float m=8,c=10, theta;

void initialize()      // draws initial house
{   glClearColor(0,0,0,0);
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();                    //sets the projection matrix as I
    gluOrtho2D(-450,450,-450,450);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();                    //sets modelview matrix as I
    glColor3f(1,0,0);
    glBegin(GL_LINE_LOOP);               // draws house
    for(int i=0;i<11;i++);
        glVertex2fv(house[i]);
    glEnd();
    glFlush();
}
```

Teacher's Signature:

```c
#include <stdio.h>
#include <GLUT/GLUT.h>
#include <math.h>

float house[11][2] = { { 100,200 },{ 200,250 },{ 300,200 },{ 100,200 },{
 100,100 },{ 175,100 },{ 175,150 },{ 225,150 },{ 225,100 },{ 300,100 },{
 300,200 } };
int angle;
float m, c, theta;

void initialize()
{
    glClearColor(0, 0, 0, 0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();  //sets to default matrix that is identity matrix
    gluOrtho2D(-450, 450, -450, 450);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glColor3f(1, 0, 0);
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < 11; i++)
        glVertex2fv(house[i]);
    glEnd();
    glFlush();
}
void display5()
{
    //ROTATED HOUSE
    glPushMatrix();
    glTranslatef(100, 100, 0);
    glRotatef(angle, 0, 0, 1);
    glTranslatef(-100, -100, 0);
    glColor3f(1, 1, 0);
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < 11; i++)
        glVertex2fv(house[i]);
    glEnd();
    glPopMatrix();
    glFlush();
    glColor3f(1, 0, 0);
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < 11; i++)
        glVertex2fv(house[i]); //sets the vertices of the house
    glEnd();
    glFlush();
}
void display52()
{
    // line
    float x1 = 0, x2 = 500;
    float y1 = m * x1 + c;
    float y2 = m * x2 + c;
 glColor3f(1, 1, 0);
 glBegin(GL_LINES);
```

EXPT. NO. NAME: Page No. Date YOUVA

```
void display1()          // print initial house
{   // draw rotated house
initialize(); glPushMatrix();
        glTranslatef(100, 100, 0);
        glRotatef(angle, 0, 0, 1);
        glTranslatef(-100, -100, 0);
        glColor3f(1, 1, 0);
        glBegin(GL_LINE_LOOP);
        for(int i=0; i<11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glFlush();
}

void display2()
  {
        initialize(); // print initial house
        //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 house
```

Teacher's Signature

EXPT. NO. | NAME: | Page No. | Date | YOUVA

```
        glPushMatrix();
        glTranslatef(0,c,0);    theta = atan(m);
        theta = theta * 180/3.14;
        glRotatef(theta, 0, 0, 1);
        glScalef(1, -1, 1);                    // Rotation about a line
        glRotatef(-theta, 0, 0, 1);
        glTranslatef(0, -c, 0);
        glBegin(GL_LINE_LOOP);
          for (int i = 0; i<11; i++)
              glVertex2fv(house[i]);
          glEnd();
          glPopMatrix();
        glFlush();
    }
void myInit()
{   glClearColor(1.0, 1.0, 1.0, 1.0);
    glColor3f(1.0, 0.0, 0.0);
    glLineWidth(2.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 500, 0, 500);
}
void mouse(int btn, int state, int x, int y)  //mouse function
{ if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) display1();
  else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) display2(); }
int main(int argc, char *argv[])
{   glutInit(&argc, argv);
    glutInitWindowSize(900,900);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    myInit();
    glutCreateWindow("House Rotation");
    initialize();
    glutDisplayFunc(display1);
    glutMouseFunc(mouse);
    glutMainLoop();
}
```

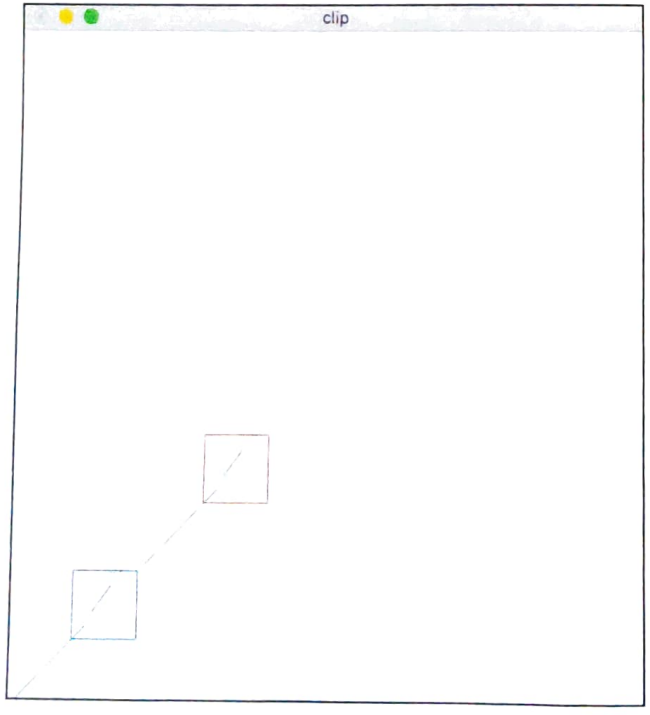
Teacher's Signature:

PGM 6

INPUT

```
Enter window coordinates (xmin ymin xmax ymax):
50 50 100 100
Enter viewport coordinates (xvmin yvmin xvmax
    yvmax) :
150 150 200 200
Enter no. of lines:
3
Enter line endpoints (x1 y1 x2 y2):
0 0 60 60
Enter line endpoints (x1 y1 x2 y2):
65 70 80 90
Enter line endpoints (x1 y1 x2 y2):
105 105 145 145
```

OUTPUT



EXPT. NO. 6

NAME: COHEN SUTHERLAND LINE CLIPPING ALGO

Page No.

Date:

PROGRAM 6

Write a program to implement the Cohen-Sutherland 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 <stdlib.h>
#include <GLUT/GLUT.h>
#define outcode int
#define true 1
#define false 0
double xmin, ymin, xmax, ymax;
double xvmin, yvmin, xvmax, yvmax;
const int RIGHT= 2;        const int BOTTOM = 4;      //code of 4 regions
const int LEFT= 1;         const int TOP = 8;
int n;
struct line_segment { int x1, y1, x2, y2; };
struct line_segment ls[10];                           // structure of line segments

outcode computeoutcode(double x, double y)            // computes code for
{    outcode code = 0;                                // each vertex
     if (y > ymax)
        code |= TOP;
     else if(y < ymin)
        code |= BOTTOM;
     if (x > xmax)
```

Teacher's Signature

```
//
//  lineclip.cpp
//  CG Lab
//
//  Created by Ruchita Biradar on 06/11/20.
//

#include <stdio.h>
#include<stdlib.h>
#include<GLUT/GLUT.h>
#include "Header.h"

#define outcode int
#define true 1
#define false 0
double xmin, ymin, xmax, 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];

outcode 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 cohensuther(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)
```

EXPT. NO. | NAME: | Page No. | Date | YOUVA

```
        code1 = RIGHT
    else if (x < xmin) code1 = LEFT;
    return code;
}

void cohensuther(double x0, double y0, double x1, double y1)
{                                          // implement Cohen Sutherland
    outcode outcode0, outcode1, outcodeout;   algo
    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 {                             // Adjusting x & y acc
            double x, y;                   // to region it lies in
            outcodeout = outcode0 ? outcode0 : outcode1;
            if (outcode & TOP)
                { x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
                  y = ymax;
                }
            else if (outcode & BOTTOM)
                { x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
                  y = ymin;
                }
            else if (outcode & 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)                          //scaling coordinates from window to viewport
    { 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);
```

Teacher's Signature:

```
        glBegin (GL_LINE_LOOP);          //plotting viewport
        glVertex2f(xvmin, yvmin);
        glVertex2F (xvmax, yvmin);
        glVertex2F (xvmin , yvmax);
        glVertex2f (xvmax, yvmax);
        glEnd ();
      glColor3f (0, 0, 1);               // plotting lines in viewport
        glBegin (GL_LINES)                 after clipping
         glVertex2d (vx0, vy0);
         glVertex2d (vx1, vy1);
         glEnd();
    }
}

void display()
{     glClear (GL_COLOR_BUFFER_BIT);
      glColor3f (0, 0, 1);
      glBegin (GL_LINE_LOOP);              // plotting Window
      glVertex2F (xmin, ymin);
      glVertex2f (xmax, ymin);
      glVertex2F (xmin, ymax);
      glVertex2F (xmax, ymax);
      glEnd();
      for (int i=0; i<n; i++)
        { glBegin (GL_LINES);              //plotting lines
          glVertex2d (ls[i].x1, ls[i].y1);   before clipping
          glVertex2d (ls[i].x2, ls[i].y2);
          glEnd();
        }
```

Teacher's Signature

EXPT. NO. | NAME | Page No. | Date | YOUVA

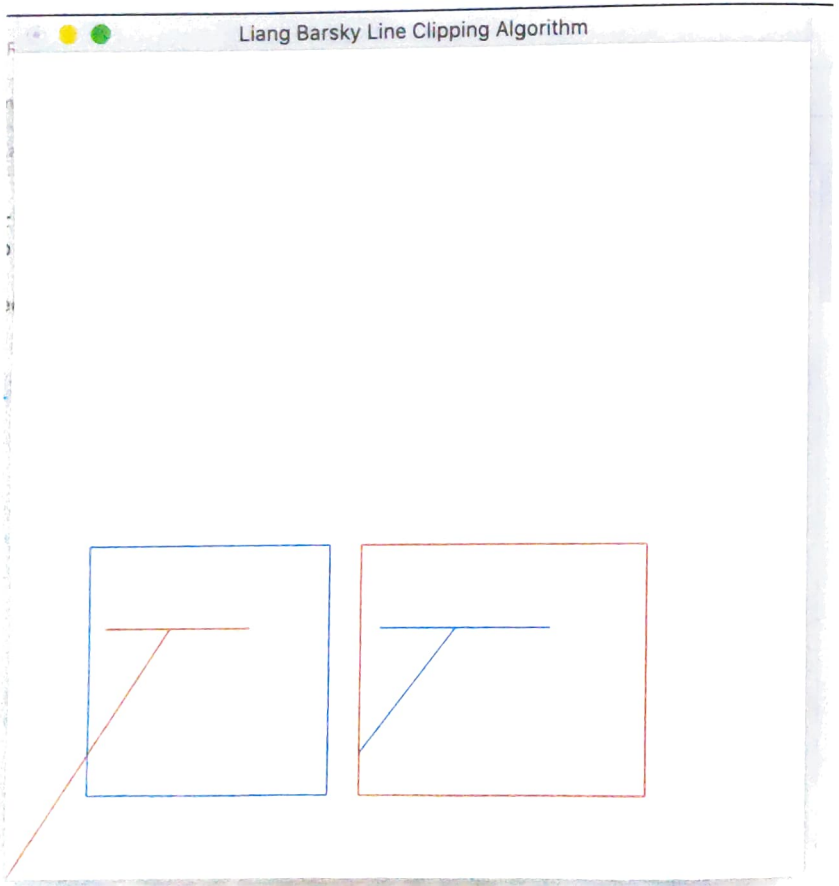
```
        for (int i = 0; i < n; i++)      //calling algo for each line segment
            cohensutherland (ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);
    glFlush();
}

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

int main(int argc, char* argv[])
{   printf("enter window coordinates\n");
    scanf("%lf %lf %lf %lf", &xmin, &ymin, &xmax, &ymax);
    printf("enter viewport coordinates\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 endpts\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);
    glutCreateWindow("Cohen Sutherland Clip");
    myinit();
    glutDisplayFunc(display);
    glutMainLoop();
}
```

Teacher's Signature

INPUT

```
Enter window coordinates: (xmin ymin xmax ymax)
50 50 200 200
Enter viewport coordinates: (xvmin yvmin xvmax yvmax)
220 50 400 200
Enter no. of lines:
2
Enter coordinates: (x1 y1 x2 y2)
0 0 100 150
Enter coordinates: (x1 y1 x2 y2)
150 150 60 150
```

OUTPUT



EXPT NO. 7 NAME: LIANG BARSKY LINE CLIPPING ALGO

Page No.:
Date:

YOUVA

PROGRAM - 7

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<GLUT/glut.h>
double xmin, ymin, xmax, ymax;          //window coordinates
double xvmin, yvmin, xvmax, yvmax;      //Viewport coordinates
int n;
struct line_segment { int x1,x2,y1,y2; };
struct line_segment ls[10];
int cliptest (double p, double q, double *u1, double *u2)
{   double r;
    if (p) r=q/p;          // to divide only if p is non zero
    if (p<0.0)             //potential entry point, update u1
      { if (r>*u1) *u1=r;
        if (r>*u2) return (false);
      }
    else if (p>0.0)        // potential exit point, update u2
      { if (r<*u2) *u2=r;
        if (r<*u1) return (false);
      }
    else if (p==0.0)
        { if (p<0.0) return (false); } //line || to edge but outside
    return (true);
}
```

Teacher's Signature:

```
//
//  Liangbarsky.cpp
//  CG Lab
//
//  Created by Ruchita Biradar on 26/11/20.
//
#include <stdio.h>
#include <GLUT/GLUT.h>
#include "Header.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* 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);
}

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, yvmin);
glVertex2f(xvmax, yvmax);
glVertex2f(xvmin, yvmax);
```

EXPT. NO. NAME: Page No. Date: YOUVA

```
void LiangBarsky (double x0, double y0, double x1, double y1);
{ double dx = x1-x0, dy = y1-y0, u1 = 0.0, u2 = 1.0;
    glColor3f(1.0, 0.0, 0.0);        // draw red Viewport
      glBegin (GL_LINE_LOOP);
        glVertex2f(xvmin, yvmin);
        glVertex2f(xvmax, yvmin);
        glVertex2f(xvmin, yvmax);
        glVertex2f( xvmax, 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 = x1 + u1*dx;
                y0 = y0 + u1 *dy;
              }
    double sx = (xvmax - xvmin)/(xmax - xmin);  // Scale parameters from
double sy = (yvmax-yvmin)/(ymax-ymin);          // Window to Viewport mappings
  double vx0 = xvmin + (x0-xmin)* sx, vy0 = yvmin + (y0-ymin)* sy;
double vx1 = xvmin + (x1-xmin) * sx, vy1 = yvmin + (y1-ymin) * sy;
        glColor3f(0.0, 0.0, 1.0);  // draw blue clipped lines
          glBegin (GL_LINES);
            glVertex2d(vx0, vy0);
            glVertex2d( vx1, vy1);
          glEnd();
}
```

Teacher's Signature

EXPT NO. | NAME | Page No. | Date | YOUVA

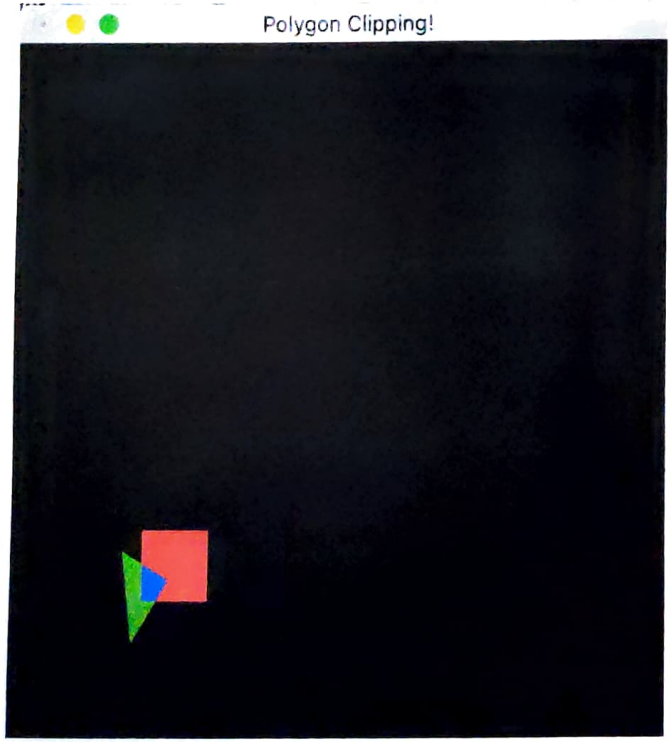
```
void display()
{   glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 0.0, 0.0);
      for (int i=0; i<n; i++)                    //draw red lines
    {  glBegin(GL_LINES); glVertex2d(ls[i].x1, ls[i].y1);
       glVertex2d(ls[i].x2, ls[i].y2); glEnd(); }
    glColor3f(0.0, 0.0, 1.0); glBegin(GL_LINE_LOOP);
glVertex2f(xmin, ymin); glVertex2f(xmax, ymin); glVertex2f(xmin, ymax); glVertex2f(xmax, ymax);
glEnd();          for(int i=0; i<n; i++)
                      LiangBarsky(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);
 glFlush();
}
void myinit()
{ glClearColor(1.0,1.0,1.0,0.0); 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 \n"); scanf("%f %f %f %f", &xmin, &ymin, &xmax, &ymax);
printf("enter viewport coordinates\n"); scanf("%f %f %f %f", &vxmin, &vymin, &vxmax, &vymax);
   printf("enter no of lines\n"); scanf("%d", &n);
          for(int i=0; i<n; i++)
           { printf("enter coordinates of line\n");
           scanf("%d %d %d %d", &ls[i].x1, &ls[i].y1, &ls[i].x2, &ls[i].y2);
           }
glutCreateWindow("Liang Barsky Clipping"); glutDisplayFunc(display);
glut.myinit();
glutMainLoop();
}
```

Teacher's Signature

INPUT

```
(base) ruchitabiradar@Ruchitas-MacBook-Pro CG Lab % ./pgm8
Enter no. of vertices:
3
Polygon Vertex:
240 230
Polygon Vertex:
170 270
Polygon Vertex:
185 140
Enter no. of vertices of clipping window:4
Clip Vertex:
200 200
Clip Vertex:
300 200
Clip Vertex:
300 300
Clip Vertex:
200 300
```

OUTPUT



EXPT NO. 8 | NAME: COHEN HODGEMAN LINE CLIPPING ALGO | Page No. | Date | YOUVA

PROGRAM - 8

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

```
#include <iostream.h>
#include <GLUT/GLUT.h>
using namespace std;
int poly_size, poly_points[20][2], org_poly_size, org_poly_points[20][2],
clipper_size, clipper_points[20][2];
const int MAX_POINTS=20;

void drawPoly( int p[][2], int n)        // draw polygon
{   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);
int den= (x1-x2)*(y3-y4) - (y1-y2)*(x3-x4);
 return num/den;          // return x value of intersection pt b/w 2 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;          // return y value of intersection pt b/w 2 lines
}
```

Teacher's Signature:

```
// C++ program for implementing Sutherland Hodgman
// algorithm for polygon clipping
#include<iostream>
#include<GLUT/glut.h>
using namespace std;
int poly_size, poly_points[20][2], org_poly_size, org_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);
    int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
    return num / den;
}

// Returns y-value of point of intersectipn 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;
}
```

EXPT. NO. | NAME: | Page No. | Date | YOUVA

```
void clip( int poly_points[][2], int & poly_size, int x1, int y1, int x2, int y2)
{ int new_points [MAX_POINTS][2], new_poly_size = 0;
  for (int i=0 ; i < poly_size ; i++)
    {  int k = (i+1)% poly_size ;     // k is next side , i is current side
       int ix = poly_points [i][0],  iy = poly_points [i][1]; // x & y of i
    int kx = poly_points [k][0], ky = poly_points [k][1]; //x & y of k
    int i_pos = (x2-x1) * (iy-y1) - (y2-y1) * (ix -x1); // position of i wrt Clipper
    int k_pos = (x2-x1) * (ky-y1) -(y2-y1) * (kx-x1) // position of k wrt Clipper
  if( i_pos>=0   && k_pos >=0)     //When both pts inside, add only 2nd pt
     { new_points [new_poly_size][0]=kx; new_points [new_poly_size][1]=ky;
       new_poly_size ++;
     }
else if( i_pos <0  && k_pos>=0) // When first point is outside, add intersection pt & 2nd pt
   { 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 ++;
   }
 else if ( i_pos >=0 && k_pos<0)  //When 2nd pt is outside, add intersection pt
  { 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 ++;
  }
}

     poly_size = new_poly_size; // copy new points into original array
     for (int i=0;i<poly_size;i++)        //change size
    { poly_points [i][0] = new_points[i][0];
      poly_points [i][1] = new_points[i][1];
    }
}
```

Teacher's Signature

EXPT NO | NAME | Page No | Date | YOUVA

```
Void init()
{  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);
}

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

int main(int argc, char* argv[])
{  printf("Enter no of vertices"); scanf("%d", &poly_size);
   if(poly_size < 3) exit(0);
     org_poly_size = poly_size;
```

Teacher's Signature

EXPT NO | NAME | Page No. | Date: | YOUVA

```
    for (int i = 0 ; i< poly_size ; i++)
{   scanf("%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("%d", &clipper_size);
    if( clipper_size<3)  exit(0);

     for (int i = 0; i < clipper_size ; i++)
       {  printf(" Enter clip vertex \n");
    scanf("%d %d", &clipper_points[i][0], &clipper_points[i][1]);
       }

    glutInit (&argc, argv);
    glutInitDisplayMode( GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (500,500);
    glutInitWindowPosition (100,100);
     glutCreateWindow ("Polygon Clipping ");
     glutDisplayFunc ( display );
     glutMainLoop ();
     return 0;
}
```

Teacher's Signature

PGM 9

OUTPUT

EXPT. NO 9 NAME: MOVING CAR USING DISPLAY LISTS Page No. Date YOUVA

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 the speed with mouse

```
#include <GLUI/GLUT.h>
#include <math.h>
#include <stdio.h>
#define CAR 1
#define WHEEL 2
float s = 1;
void carlist ()
{   glNewList ( CAR, GL_COMPILE);          //modelling the car
    glColor3f( 1, 1, 0);
    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();
    glFlush(); glEndList();
}

void wheellist ()
{   glNewList (WHEEL, GL_COMPILE_AND_EXECUTE);
```

Teacher's Signature

```c
#include<GLUT/glut.h>
#include<math.h>
#include<stdio.h>
#include "Header.h"
#define CAR 1
#define WHEEL 2
float s = 1;
void carlist() {
    glNewList(CAR, GL_COMPILE);
    glColor3f(1, 1, 0);
    glBegin(GL_POLYGON);
    glVertex3f(0, 25, 0);
    glVertex3f(90, 25, 0);
    glVertex3f(90, 55, 0);
    glVertex3f(80, 55, 0);
    glVertex3f(20, 75, 0);
    glVertex3f(0, 55, 0);
    glEnd();
    glEndList();
}
void wheellist() {
    glNewList(WHEEL, GL_COMPILE_AND_EXECUTE);
    glColor3f(1, 0, 0);
    glutSolidSphere(10, 25, 25);
    glEndList();
}
void mykeyboard(unsigned char key, int x, int y) {
    switch (key) {
    case 't': glutPostRedisplay();
        break;
    case 'q': exit(0);
    default: break;

    }
}

void myInit() {
    glClearColor(0, 0, 0, 0);
    glOrtho(0, 600, 0, 600, 0, 600);

}
void draw_wheel() {
    glColor3f(1, 0, 0);
    glutSolidSphere(10, 25, 25);

}
```

EXPT. NO. | NAME: | Page No. | Date | YOUVA

```c
    glColor3f(1,0,0);                    // modelling the wheels
    glutSolidSphere(10,25,25);
    glEndList();
}

Void mykeyboard(unsigned char key, int x, int y)
{   switch(key)
    { case 't': glutPostRedisplay();     // keyboard functionality
                break;                   // to move forward or
      case 'q': exit(0);                 // quit
      default : break;
    }
}

void myinit()
{   glClearColor(0,0,0,0);
    glOrtho(0,600,0,600,0,600);
}

void draw_wheel()
{   glColor3f(1,0,0);
    glutSolidSphere(10,25,25);
}

Void movecar(float s){
  glTranslatef(s,0.0,0.0);
  glCallList(CAR);
  glPushMatrix();
```

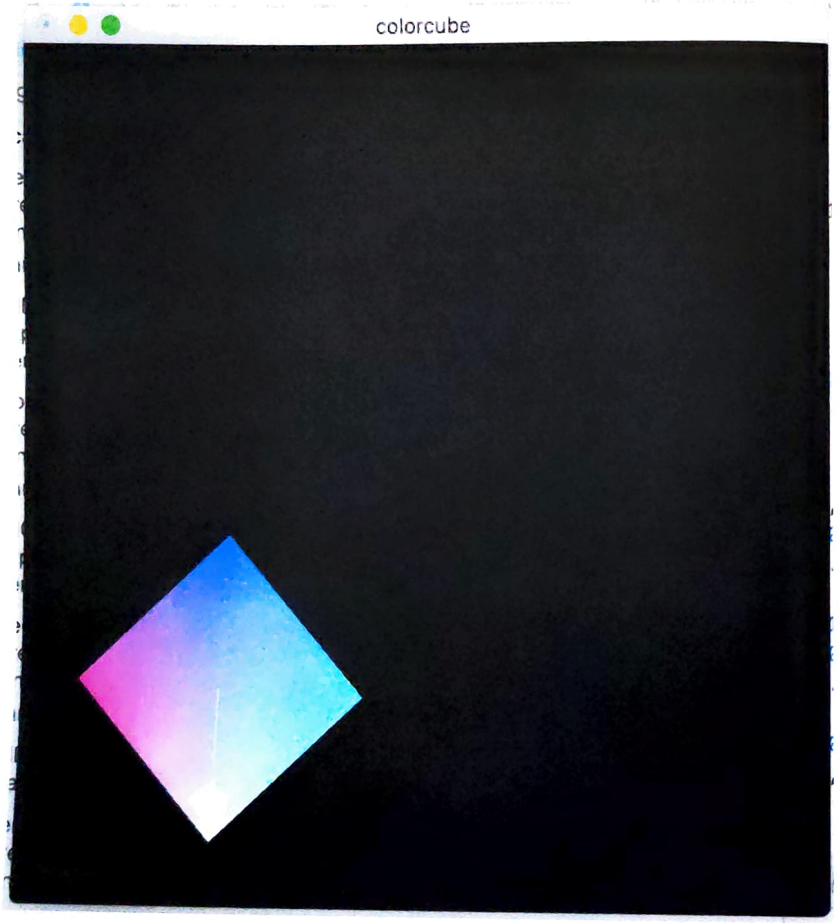
Teacher's Signature

```
        glTranslatef(25,25, 0.0);   // move 1st wheel position
        draw_wheel();
        glCallList(WHEEL);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(75, 25, 0.0);   // move 2nd wheel position
        draw_wheel();
        glCallList(WHEEL);
        glPopMatrix();
        glFlush();
    }
void myDisp(){
    glClear(GL_COLOR_BUFFER_BIT);
    carInit();
    moveCar(s);
    wheelInit();
}
void mouse(int btn, int state, int x, int y)
{ if(btn==GLUT_LEFT_BUTTON && state==GLUT_DOWN) {s+=7; myDisp();}
  else if(btn==GLUT_RIGHT_BUTTON && state==GLUT_DOWN){s+=1; myDisp();}
}
int main(int argc, char* argv[])
{   glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow("Car");
    myInit();
    glutDisplayFunc(myDisp); glutMouseFunc(mouse);
    glutKeyboardFunc(myKeyboard); glutMainLoop();
}
```

Teacher's Signature:

PGM 10

OUTPUT



EXPT. NO. 10 NAME: COLOR CUBE SPIN Page No.: Date: YOUVA

PROGRAM - 10

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

```
#include <stdlib.h>
#include <GLUT/glut.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;
void delay(float secs)
{ float end=clock()/CLOCKS_PER_SEC+secs;
  while((clock()/CLOCKS_PER_SEC)<end);
}
void displaySingle(void)                    //display callback
{ glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glRotatef(theta[0],1.0,0.0,0.0);          //rotate cube
  glRotatef(theta[1],0.0,1.0,0.0);
  glRotatef(theta[2],0.0,0.0,1.0);
```

Teacher's Signature

```
//  Created by Ruchita Biradar on 02/12/20.
//

#include <stdlib.h>
#include <GLUT/GLUT.h>
#include<OpenGL/GL.h>
#include<OpenGL/GLU.h>
#include <time.h>
#include "Header.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;

void delay(float secs)
{
    float end = clock() / CLOCKS_PER_SEC + secs;
    while ((clock() / CLOCKS_PER_SEC) < end);
}

void displaySingle(void)
{

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    glRotatef(theta[0], 1.0, 0.0, 0.0);
    glRotatef(theta[1], 0.0, 1.0, 0.0);
    glRotatef(theta[2], 0.0, 0.0, 1.0);

    glDrawElements(GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);

    glBegin(GL_LINES);
    glVertex3f(0.0, 0.0, 0.0);
    glVertex3f(1.0, 1.0, 1.0);
    glEnd();

    glFlush();
}
```

```
    glDrawElements ( GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);
    glBegin ( GL_LINES);                    //draw cube
     glVertex3f (0.0, 0.0,0.0);
     glVertex3f (1.0, 1.0,1.0);
     glEnd ();
    glFlush ();
}
void spinCube ()   // Idle callback, spin cube 2° about selected axis
{     delay (0.01);
      theta [axis] += 2.0 ;
      if (theta[axis] > 360.0)  theta [axis] -= 360.0;
      glutPostRedisplay ();
}
void mouse (int btn, int state, int x, int y){
 if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)   axis = 0;
 if (btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)  axis = 1;
 if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)  axis = 2;
}                                   //selecting axis about which to rotate

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);
```

Teacher's Signature:

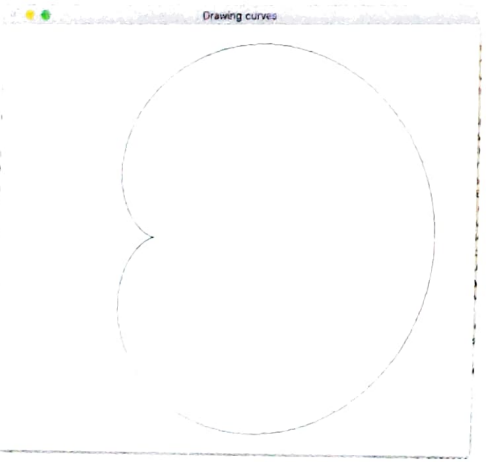
EXPT NO | NAME: | Page No.: | Date: | YOUVA

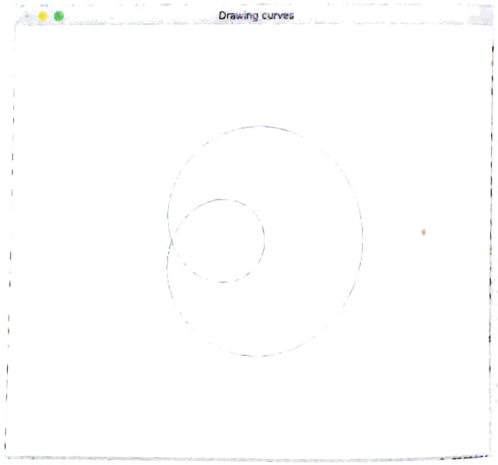
```
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char * argv[])
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowPosition(100,100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("Color Cube");
    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();
}
```

Teacher's Signature

PGM 11

OUTPUT





EXPT. NO. 11 NAME: CURVE CREATION MENU Page No.: Date: YOUVA

PROGRAM - 11

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

```
#include <GLUT/GLUT.h>
#include <math.h>
#include <stdio.h>
struct screenPt { int x, y; };
typedef enum {limacon = 1, cardiod = 2, threeleaf = 3, spiral = 4} curveName;
int w = 600, h = 500;
int curve = 1, red = 0, green = 0, blue = 0;
void lineSegment (screenPt p1, screenPt p2)   // draws line
{   glBegin(GL_LINES);
    glVertex2i (p1.x, p1.y);
    glVertex2i (p2.x, p2.y);
    glEnd();
    glFlush();
}
void drawCurve( int curveNum)                 // draws curve
{ const double twopi = 6.28185;
  const int a = 175, b = 60;
  float r, theta, dtheta = 1.0/ float(a);
  int x0 = 200, y0 = 250;
```

Teacher's Signature

```
#include<GLUT/GLUT.h>
#include<math.h>
#include<stdio.h>
#include "Header.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 myinit11(void) {
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}
void lineSegment(screenPt p1, screenPt p2) {

    glBegin(GL_LINES);
    glVertex2i(p1.x, p1.y);
    glVertex2i(p2.x, p2.y);
    glEnd();
    glFlush();
}
void drawCurve(int curveNum) {
    const double twoPi = 6.283185;
    const int a = 175, b = 60;
    float r, theta, dtheta = 1.0 / float(a);
    int x0 = 200, y0 = 250;
    screenPt curvePt[2];
    curve = curveNum;
    glColor3f(red, green, blue);
    curvePt[0].x = x0;
    curvePt[0].y = y0;
    glClear(GL_COLOR_BUFFER_BIT);
    switch (curveNum) {
    case limacon: curvePt[0].x += a + b; break;
    case cardioid: curvePt[0].x += a + a; break;
    case threeLeaf: curvePt[0].x += a; break;
    case spiral: break;
    default: break;
    }
    theta = dtheta;
    while (theta < twoPi) {
```

EXPT. NO. | NAME | Page No. | Date | YOUVA

```
screenPt[0].x = x0; screenPt curvePt[2];
screen{                curve = CurveNum;
glColor3f( red, green, blue);
curvePt[0].x = x0;   curvePt[0].y = y0;
glClear (GL_COLOR_BUFFER_BIT);
switch (CurveNum)                          //sets x value
{  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;  //sets r value
     case cardioid : r = a*(1+ cos(theta)); break;
     case threeleaf : r = a*cos(3*theta); break;
     case spiral : r = (a/4.0) *theta; break;
     default : break;
   }
   curvePt[1].x = x0 + r* cos(theta);          //sets y value
   curvePt[1].y = y0 + r* sin(theta);
   lineSegment( curvePt[0], curvePt[1]);
   curvePt[0].x = curvePt[1].x;                //last line from point 1
   curvePt[0].y = curvePt[1].y;
   theta += dtheta;
}
}
```

Teacher's Signature

EXPT. NO. | NAME: | Page No.: | Date: | YOUVA

```
void colorMenu (int id)
  {   switch(id)
      { case 0: break;                                    //sets color
        case 1: red=0; green=0; blue=1; break;
        case 2: red=0; green=1; blue=0; break;
        case 3: red=0; green=1; blue=1; break;
        case 4: red=1; green=0; blue=0; break;
        case 5: red=1; green=0; blue=1; break;
        case 6: red=1; green=1; blue=0; break;
        case 7: red=1; green=1; blue=1; break;
        default: break;
      }
      drawCurve(curve);
  }
void main_menu(int id)
  {  switch(id)
     {  case 3: exit(0);
        default: break;
     }
  }
void mydisplay()
{ glClear(GL_COLOR_BUFFER_BIT); }
void myreshape(int nw, int nh)
{ glMatrixMode(GL_PROJECTION);
  glLoadIdentity(); gluOrtho2D(0.0,(double)nw, 0.0,(double)nh);
}
void myinit(void) { glClearColor(1.0,1.0,1.0,1.0);
 glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}
```

Teacher's Signature

EXPT NO. | NAME | Page No.: | Date: | YOUVA

```
int main (int argc, char *argv[])
{ glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (w, h);
  glutInitWindowPosition (100, 100);
  glutCreateWindow ("Curve Menu");                    // choosing curve
  int curveId = glutCreateMenu (drawCurve);
   glutAddMenuEntry ("Limacon", 1);
   glutAddMenuEntry ("Cardioid", 2);
   glutAddMenuEntry ("Threeleaf", 3);
   glutAddMenuEntry ("Spiral", 4);
   glutAttachMenu (GLUT_LEFT_BUTTON);
   int colorId = glutCreateMenu (colorMenu);          // choosing color
   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);             // main menu
   glutAddSubMenu ("colors", colorId);
   glutAddMenuEntry ("quit", 3);
  glutAttachMenu (GLUT_LEFT_BUTTON); myinit();
   glutDisplayFunc (mydisplay); glutReshapeFunc (myreshape);
   glutMainLoop ();
}
```

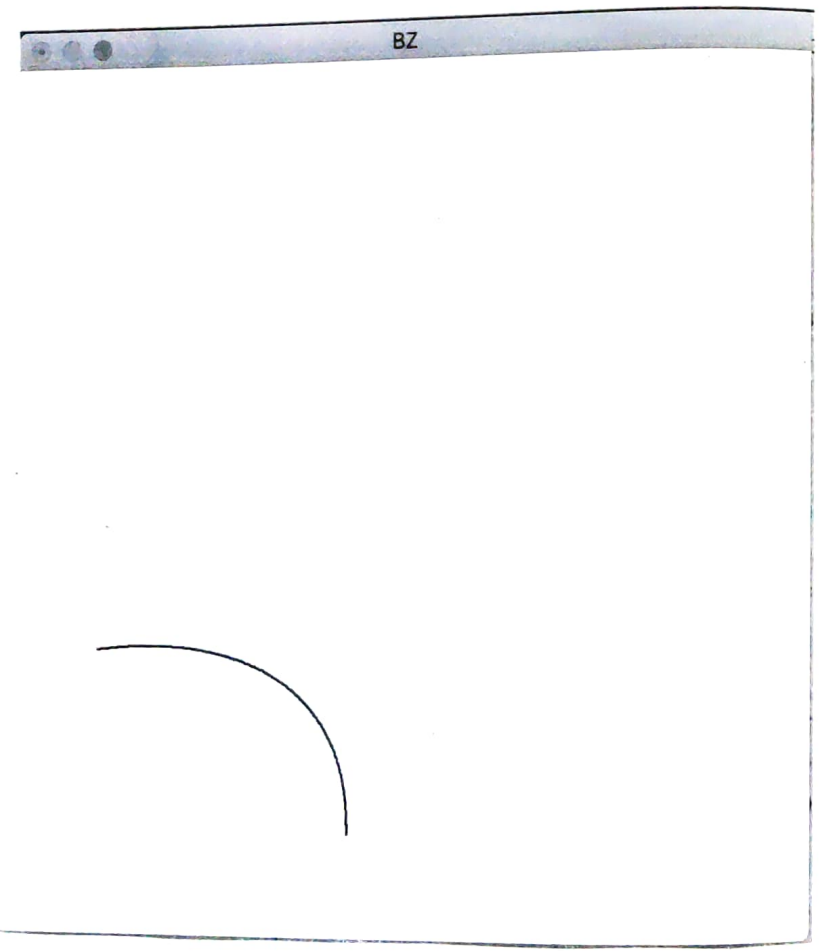
Teacher's Signature

PGM 12

INPUT

```
X: 113 Y323 X: 295 Y347 X: 440 Y282 X: 436 Y115
```

OUTPUT



EXPT NO 12 NAME: BEZIER CURVE Page No. Date YOUVA

PROGRAM - 12

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

```
#include <iostream.h>
#include <math.h>
#include <GLUT/GLUT.h>
using namespace std;
float f, g, x, x1[4], yc[4];
int flag = 0;
void myinit()                          //initialising function
{ glClearColor(1,1,1,1);
  glColor3f(1,1,1);
  glPointSize(5);
  gluOrtho2D(0,500,0,500);
}
void drawPixel(float x, float y)       //to draw points
{ glBegin(GL_POINTS);
  glVertex2f(x,y);
  glEnd();
}
void display()
{ glClear(GL_COLOR_BUFFER_BIT);
  int i; double t;
  glColor3f(0,0,0);
  glBegin(GL_POINTS)                   // draw points between
   for(t=0; t<1; t=t+0.005)            the 4 specified points
```

Teacher's Signature:

```
#include<math.h>
#include<GLUT/GLUT.h>

using namespace std;
float f, g, r, x1[4], yc[4];
int flag = 0;
void myInit12() {

    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 display12() {

    glClear(GL_COLOR_BUFFER_BIT);
    int i;
    double t;
    glColor3f(0, 0, 0);
    glBegin(GL_POINTS);
    for (t = 0; t < 1; t = t + 0.005) {
        double xt = pow(1 - t, 3) * x1[0] + 3 * t * pow(1 - t, 2) * x1[1] +
         3 * pow(t, 2) * (1 - t) * x1[2] + pow(t, 3) * x1[3];
        double yt = pow(1 - t, 3) * yc[0] + 3 * t * pow(1 - t, 2) * yc[1] +
         3 * pow(t, 2) * (1 - t) * yc[2] + pow(t, 3) * yc[3];
        glVertex2f(xt, yt);

    }
    glColor3f(1, 1, 0);
    for (i = 0; i < 4; i++) {
        glVertex2f(x1[i], yc[i]);
        glEnd();
        glFlush();
    }


}
void mymouse12(int btn, int state, int x, int y)
{
    if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN && flag < 4)
    {
        x1[flag] = x;
        yc[flag] = 500 - y;
        cout << " X: " << x << " Y" << 500 - y;
        glPointSize(3);
        glColor3f(1, 1, 0);
```

EXPT NO | NAME: | Page No. | Date | YOUVA

```
{ double xt = pow(1-t,3)*x1[0] + 3*t *pow(1-t,2)* x1[1] +
        3*pow(t,2)*(1-t)* x1[2] + pow(t,3) * x1[3];
double yt = pow(1-t,3)*yc[0] + 3*t * pow(1-t,2) *yc[1]+
        3 * pow(t,2) * (1-t) * yc[2] + pow(t,3) *yc[3];
}
glColor3f(1,1,0);
for(i=0; i<4; i++)
    { glVertex2f(x1[i], yc[i]); }      // plot the 4 given points
    glEnd(); glFlush();                  in yellow
}

int main (int argc, char*argv[])
{   glutInit (&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    glutInitWindowPosition (0,0);
    glutCreateWindow (" Bezier Curve");
    glutDisplayFunc (display);
    myinit();
    glutMainLoop();
    cout << " Enter x coordinates ";
    cin >> x1[0] >> x1[1] >> x1[2] >> x1[3];
    cout << " Enter y coordinates ";
    cin >> yc[0] >> yc[1] >> yc[2] >> yc[3];
}
```

Teacher's Signature: