Write a program to generate a line using Bresenham's line drawing technique. Consider slopes greater and lesser than one.

```
#include <iostream> ~~#include~~
#include <GL/glut.h>
#include <time.h>
using namespace std;
int x1, x2, y1, 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, incx, incy, inc1, inc2, 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) { 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;
y+=incy; draw_pixel(x,y);
  } }  glFlush(); }
void myinit(){
      glClear(GL_COLOR_BUFFER_BIT);
      glClearColor(1,1,1,1);
      gluOrtho2D(-250, 250, -250, 250); }
void MyMouse(int button, int state, int x, int y){
  switch (button){
  case GLUT_LEFT_BUTTON: if (state == GLUT_DOWN){
        if (flag==0){ printf("defining x1,y1");
        x1=x-250;  yc1=250-y;
        flag++;   cout<< x1<<" " <<yc1 << "\n";
        } else {
              printf("defining x2,y2");
        x2=x-250; y2=250-y; flag=0;
        cout<<x2<<  "  " <<y2<<"\n";
        draw_line(); } } break; } }
Void display(){ }
int main (int ac, char *av[]){
        glutInit(&ac,av);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(100, 200);
        glutCreateWindow("LINE");
        myInit();
        glutMouseFunc(MyMouse); // include to use mouse
         //draw_line(); //include to use keyboard
        glutDisplayFunc(display);
        glutMainLoop();
      }
```

Write a program to generate a circle and ellipse using Bresenham's circle drawing and ellipse drawing techniques.

```
#include<gl/glut.h>
#include <stdio.h>
#include <math.h>
int xc, yc, r, 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, 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; point1_done=0;
    void myMouseFunc circle (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;
                }
```

Teacher's Signature

```
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)^2;
    r=(int)(sqrt(exp));
    circlebres(); point1_done=0;
    }}
//ELLIPSE
void draw_ellipse(int xce,int yce, int x ,int y){
    glBegin(GL_POINTS);
    glVertex2i(x+xce, y+yce); glVertex2i(x-xce, y+yce);
    glVertex2i(x+xce,-y+yce); glVertex2i(-x+xce,-y+yce);
    glEnd(); }
void midptellipse(){ glClear(GL_COLOR_BUFFER_BIT);
    float dx, dy, d1, d2, x, y;
      x=0; y=ry;
  //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;
   while (dx<dy){                          // for region 1
            draw_ellipse(xce,yce,x,y);     // 4-way symmetry
          if (d1<0){ x++;                  // update decision parameter
            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 for region 2
  d2=((ry*ry)*(x+0.5)*(x+0.5))) + ((rx*ry)*(y-1)^2)
        -(rx*rx*ry*ry);
```

```
// Plotting points of region 2
while(y >= 0) { draw_ellipse (xce, yce, x, y);
    //checking & updating parameter value based on alg
    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; 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 - p1e_x)*(p2e_x - p1e_x) + (p2e_y - p1e_y)^2;
            rx = (int)(sqrt (exp));
            pointle_done = 2; }  ~~//midptellipse()~~
    else if (button== GLUT_LEFT_BUTTON && state == GLUT_DOWN &&
      pointle_done=2) { p3e_x = x-250;  p3e_y= 250-y;
    float exp = ( p3e_x - p1e_x) + (p3e_x - p1e_x) + (p3e_y - p1e_y)^2;
            ry = (int )(sqrt(exp));
            midptellipse ();
            pointle_done = 0; } }
```

```
void myDrawing() {}
void myDrawingc() {}
void minit() {  glClearColor(1,1,1,1);
                glColor3f(1,0,0);
                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 KEYBOARD
      printf("Enter 1 to draw circle, 2 to draw ellipse\n");
      int ch;
      scanf("%d",&ch);
      switch(ch){ case 1:
                    printf("Enter centre of circle and radius\n");
                  scanf("%d%d%d%d", &xc, &yc, &r);
                  glutCreateWindow("Circle");
                  glutDisplayFunc(circlebres); break;
      Case 2: printf("Enter centre of ellipse, major & minor radius\n");
            scanf("%d%d%d%d", &xce, &yce, &rx, &ry);
                  glutCreateWindow("Ellipse");
                  glutDisplayFunc(midptellipse); break;
                  }
            //END KEYBOARD
            minit();
            glutMainLoop();
            }
```

Write a program to recursively subdivide a tetrahedron to form 3D Sierpinski gasket. No. of steps to be specified at execution.

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

int m;
typedef float point[3];
point tetra[4] = { {0,100,-100}, {0,0,100}, {100,-100,-100},
                   {-100,-100,-100} };
void tetrahedron (void);
void myinit (void);
void divide_triangle( point a, point b, point c, int m);
void draw_triangle (point p1, point p2, point p3);
int main (int argv, char ** argc) {
        printf ("enter no. of iterations \n");
        scanf_s("%d", &m);
        glutInit (&argv, argc);
        glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowPosition (100, 200);
        glutInitWindowSize (500, 500);
        glutCreateWindow ("Serpinski 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 (-500,500,-500,500,-500,500); }
void tetrahedron () { ~~glClearColor~~
        glClear (GL_COLOR_BUFFER_BIT | GL_~~COLOR~~ DEPTH_BUFFER_BIT);
        glColor3f(1,0,0);
        divide_triangle(tetra[0], tetra[1], tetra[2], m);
        glColor3f (0,1,0);
        divide_triangle (tetra[3], tetra[2], tetra[1], m);
        glColor3f (0,0,1); divide_triangle(tetra[0], tetra[3], tetra[1], m);
        glColor3f (0,0,0); divide_triangle (tetra[0], tetra[2], tetra[3], m);
        glFlush(); }

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

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

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

void 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], sl); }
sort (intx, (intx+m));
if (m >=2)
  for (int i=0; i<m; i+=2)
    draw_line(intx[i], sl, intx[i+1], sl); } }
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);
  //glFlush();      }
void myInit() { glClearColor(1,1,1,1);
                glColor3f(0,0,1);
                glPointSize(1);
                gluOrtho2D(0, wx, 0, wy); }
void main (int ac, char* av[]) { glutInit(&ac, &av);
  printf("Enter no of sides:\n"); scanf("%d", &n);
  printf("Enter coordinates of endpoints:\n");
  for(int i=0; i<n; i++) { printf("X-coord  Y-coord\n");
                           scanf("%f %f", &x[i], &y[i]); }
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("scanline");
  glutDisplayFunc(display_filled_polygon);
  myInit();
  glutMainLoop();
}
```

Write program to create a house figure and perform:

(i) Rotate it about given fixed point using OpenGL transformation

(ii) Reflect it about an axis: $y=mx+c$ using OpenGL transformation

```
#include <gl/glut.h>
#include <math.h>
// #include <stdlib.h>
#include <stdio.h>
float house[11][2]={{100,200},{200,250},{300,200},{100,200},
 {100,100},{175,100},{225,150},{225,100},{300,100},{300,200}};
int angle; float m, c, theata;
void display (){ glClearColor(1,1,1,0);
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-450,450,-450,450);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
//normal house
        glColor3f(1,0,0);
          glBegin(GL_LINE_LOOP);
        for (int i=0;i<11;i++)
          glVertex2fv(house[i]);
          glEnd();glFlush();
//rotated house
        glPushMatrix();
        glTranslatef(0,c,0);
         theta = atan(m);
        theta = theta * 180/3.14;
        glRotatef(theta,0,0,1);
        glScalef(1,-1,1);
```

Teacher's Signature

```
glRotatef (-theta,0,0,1);
glTranslate (0,-c,0);
glBegin(GL_LINE_LOOP);
for(int i=0; i<11; i++) glVertex2fv (house[i]);
glEnd(); glPopMatrix();
        glFlush(); }
void display2(){ glClearColor(1,1,1,0);
                glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFF.)
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
//normal house
    glColor3f(1,0,0);
    glBegin(GL_LINE_LOOP);
  for(int i=0; i<11; i++) glVertex2fv(house[i]);
    glEnd(); glFlush();
//line
  float x1=0, x2=500, y1= m*x1+c, 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(int i=0; i<11; i++) glVertex2fv(house[i]);
    glEnd(); glPopMatrix(); glFlush();  }
void myInit() { glClearColor(1,1,1,1);
        glColor3f(1,0,0);
        glLineWidth(2);
        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 rotation angle \n");
            scanf ("%d", &angle);
            printf ("Enter c, 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();
        }
```

Write a program to implement Cohen Sutherland line clipping algorithm. Make provision to specify input for multiple lines.

```
#include <stdio.h>
#include <stdlib.h>
#include <gl/glut.h>
~~double x m~~
#define outcode int
#define true 1
#define false 0
double xmin, ymin, xmax, ymax, xvmin, yvmin, xvmax, yvmax;
const int RIGHT=4, LEFT=8; TOP=1, 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) done=true;
else { double x, y;
       outcodeout = outcode0? outcode0 : outcode1;
       if(outcodeout & TOP){ y = ymax;
           x = x0 + (x1-x0) * (ymax-y0)/(y1-y0); }
       else if (outcodeout & BOTTOM){ y = ymin;
           x = x0 + (x1-x0) * (ymin - y0) / (y1-y0); }
       else if (outcodeout & RIGHT){ x = xmax;
           y = y0 + (y1-y0) * (xmax - x0)/(x1-x0); }
       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){
    double sx=(xvmax - xvmin)/(xmax - xmin);
    double vx0= xvmin + (x0 - 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 (0,0,0); glBegin(GL_LINES);
```

```
    glVertex2d(vx0, vy0); glVertex2d(vx1, vy1);
    glEnd();
  }}
void display() { glClear(GL_COLOR_BUFFER_BIT);
                glBegin(GL_LINE_LOOP);
    glVertex2f(xmin, ymin); glVertex2f(xmax, ymin);
    glVertex2f(xmax, ymax); glVertex2f(xmin, ymax);
                glEnd();
    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(); }
    for (int i=0; i<n; i++)
            cohensuther(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);
    glFlush(); }
void main (int argc, char** argv){
  printf("Enter window xmin, ymin, xmax, ymax \n");
  scanf("%lf %lf %lf %lf", &xmin, &ymin, &xmax, &ymax);
  printf("Enter viewport xvmin, yvmin ... \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 x1, x2, y1, y2 \n")
                scanf("%d %d %d %d", &ls[i].x1, &ls[i].x2, y2)
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("clip");
  myInit();
  glutDisplayFunc(display);
  glutMainLoop(); }
```

#Write a program to implement Liang-Barsky line clipping algorithm Make provision to specify input for multiple lines.

```
#include <stdio.h>
#include <GL/glut.h>
double xmin, ymin, xmax, ymax, xvmin, yvmin, xvmax, yvmax;
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 if p
     if (p< 0.0) {                    // potential entry point, update te
          if (r> *u1) *u1 = r;
          if (r > *u2) return(false); } // line portion outside
     else if (p >0.0) {                // potential leaving point, up tl
          if (r< *u2) *u2 = r;
          if (r < * u1) return (false); } // line portion outside
     else if (p==0.0){
          if(q< 0.0) return (false); }     // parallel to edge but outside
       return (true)                                  }
Void LiangBarskyClipandDraw (double x0, double y0, double x1, double y1){
    double dx = x1 - x0, dy = y1 - y0, u1 = 0, u2 = 1;
    glColor3f (1,0,0);                          // red colored viewport
    glBegin (GL_LINE_LOOP);
     glVertex2f(xvmin, yvmin); glVertex2f(xvmax, yvmin),
     glVertex2f(xvmax, yvmax); glVertex2f(xvmin, yvmax);
     glEnd();
```

```
if(cliptest (-dx, x0-xmin, &u1, &u2)        // inst wrt left edge
if(cliptest (dx, xmax-x0, &u1, &u2)         // inside wrt right edge
if(cliptest (-dy, y0-ymin, &u1, &u2)        // inside wrt bottom edge
if cliptest (dy, ymax-y0, &u1, &u2){        // wrt top edge
    if (u2<1.0) { x1=x0+u2*dx;
                  y1=y0+u2*dy; }
    if(u1>0) { x0= x0+u1*dx;
               y0 = y0+u1 * dy; }
double sx= (xvmax-xvmin)/(xmax-xmin);       // window to viewport
double sy = (yvmax-yvmin)/(ymax-ymin);      //scale parameters
double vx0 = xvmin + (x0-xmin)* sx, vy0= yvmin + (y0-ymin)*sy,
  vx1= xv min + (x1-xmin)*sx, vy1= yvmin +(y1-ymin)*sy;
glColor3f (0,0,1)                           //blue colored clipped line)
 glBegin (GL_LINES);
 glVertex2d(vx0, vy0); glVertex2d (vx1, vy1);
  glEnd(); } }                              //end of line clipping
void display() { glClear (GL_COLOR_BUFFER_BIT);
      glColor3f(1,0,0);
   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();          }
      glColor3f(0,0,1);
      glBegin (LINE_LOOP);
         glVertex2f(xmin, ymin); glVertex2f(xmax, ymin);
         glVertex2f(xmax, ymax); glVertex2f( xmin, ymax);
      glEnd();
   for(int i=0; i<n; i++)
         LiangBarskyClipandDraw (ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);
   glFlush();
   }
```

```
void myinit(){ glClearColor(1,1,1,1);
                glColor3f (1,0,0); glLineWidth(2.0);
                glMatrixMode (GL_PROJECTION);
                glLoadIdentity(); gluOrtho2D(0,499,0,499); }
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 ");
        scanf ("%lf %lf %lf %lf", &xmin, &ymin, &xmax, &ymax);
        printf ("Enter viewport : xvmin, yvmin, xvmax, yvmax ");
        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 ("coordinates: x1 y1 x2 y2\n");
                scanf ("%d %d %d %d, &ls[i].x1, &ls[i].y1, &ls[i].x2, y2);
                }
        glutCreateWindow ("Liang Barsky line clipping algorithm");
        glutDisplayFunc (display);
         myinit();
        glutMainLoop();
        }
```

Write a program to implement Cohen Hodgeman polygon clipping algorithm. Make provision to specify input polygon.

```
#include <iostream>
#include <GL/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)*(y3-y4) - (y1-y2)*
                (x3*y4 - y3*x4);
    int den= (x1-x2)*(y3-y4) - (y1-y2)*(x3-x4);
        return num/den; }
// clips all edges wrt one clip edge of clipping area
void clip(int poly_points[][2], int poly_size, int x1, int y1, x2,y2){
    int new_points[MAX_POINTS][2], new_poly_size=0;
// (ix, iy), (kx,ky) are coordinate values of the points
    for (int i=0; i<poly_size; i++){
        int k = (i+1) % poly_size;
    int ix = poly_points[i][0], iy = poly_points[i][1];
    int kx = poly_points[k][0], ky = poly_points[k][1];
    int i_pos= (x2-x1)*(iy-y1) -(y2-y1)*(ix-x1);
    int k_pos= (x2-x1)*(ky-y1) -(y2-y1)*(kx-x1);
```

```
if (i_pos>=0 && k_pos >=0){          // case 1: both points inside
    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){          //case 2: first point outside
    new_points [new_poly_size][0] = x_intersect(x1,y1,x2,y2,ix,iy,kx,ky);
    new_points [new_polysize][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) {          //case 3: only 2nd pt outside
    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++; }
else {
    // case 4: both points outside, no points added
    }
    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 (0,0,0,0);
              glMatrixMode (GL_PROJECTION);
              glLoadIdentity ();
              glOrtho (0,500,0,500,0,500);
              glClear (GL_COLOR_BUFFER_BIT); }
```

```
void display(){ init();
        glColor3f(1,0,0);
        drawPoly(clipper_points, clipper_size);
        glColor3f(0,1,0);
        drawPoly(org_poly_points, org_poly_size);
    for(int i=0; i< clipper_size; i++){
        int k = (i+1)% clipper_size;      //i,k two consecutive indices
    clip(poly_points, poly_size, clipper_points[i][0],
clipper points[i][1], clipper_points[k][0], clipper_points[k][1]);}
        glColor3f(0,0,1);
        drawPoly(poly_points, poly_size);
        glFlush();  }
int main(int argc, char * argv[]){
    printf("Enter no. of vertices");
    scanf("%d", &poly_size);
    org_poly_size = poly_size;
    for(int i=0; i< poly_size; i++){
      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("clipvertex\n");
    scanf_s("%d%d", &clipper_points[i][0], &clipper_points[i][1]);}
    glutCreateWindow("Polygon Clipping");
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Write a program to model a car like figure using display lists and move from one end of screen to other end. User control speed with mous

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

void wheellist(){
        glNewList (WHEEL, GL_COMPILE_AND_EXECUTE);
        glColor3f (0,1,1);
        glutSolidsphere (10,25,25);
        glEndList(); }
void mykeyboard(unsigned char key,int x, int y){
    switch(key) {
    case '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(0,1,1);
                    glutSolidSphere(10, 25, 25); }

void moveCar(float s){  glTranslatef(s, 0, 0);
                        glCallList(CAR);
                        glPushMatrix();
                        glTranslatef(25, 25, 0);  //move to 1st wheel pos
    //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=GLUT_DOWN){
          s+=2; myDisp();                              }
                                                 }
```

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

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

```
#include <stdlib.h>
#include <GL/glut.h>
#include <gl\GL.h>
#include <gl\GLU.h>
#include <time.h>
GLfloat vertices[]= {-1,-1,-1,1,-1,-1,1,1,-1,-1,1,-1,-1,-1,
    1,1,-1,1,1,1,1,-1,1,1}
GLfloat normals[]= {-1,-1,-1,1,-1,-1,1,1,-1,-1,1,-1,-1,-1,
    1,1,-1,1,1,1,1,-1,1,1}
GLfloat colors[]= {0,0,0,1,0,0,1,1,0,0,1,0,0,0,1,1,
    0,1,1,1,1,0,1,1}
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};
static GLfloat beta[]={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, clear frame buffer & z buffer, rotate cube and draw */
  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  glRotatef(theta[0],1,0,0);
  glRotatef(theta[1],0,1,0);
  glRotatef(theta[2],0,0,1);
```

```
glDrawElements (GL_QUADS, 24, GL_UNSIGNED_BYTE, cubeIndices);
glBegin(GL_LINES);
glVertex3f(0,0,0);
glVertex3f(1,1,1);
glEnd(); glFlush; }

void spinCube() {  //spin cube abt 2 degrees selected axis
    delay(0.01);
    theta[axis]+=2.0;
    if (theta[axis] > 360) theta[axis]-=360;
    glutPostRedisplay(); }

void mouse(int btn, int state, int x, int y){
    if (btn== GLUT_LEFT_BUTTON && state== GLUT_DOWN) axis=0;
    if (btn== GLUT_MIDDLE_BUTTON && state= = GLUT_DOWN) axis=1;
    if(btn == GLUT_RIGHT_BUTTON && state= = GLUT_DOWN) axis= 2; }

void myReshape(int w, int h){
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    if(w<=h)
glOrtho(-2, 2, -2 * (GLfloat)h/(GLfloat)w, 2 * (GLfloat)h/(GLfloat)w, -10, 10);
    else { glOrtho(-2 * (GLfloat)w/(GLfloat)h, 2 * (GLfloat)w/(GLfloat)h,
-2, 2, -10, 10); }
    glMatrixMode(GL_MODELVIEW);
}
```

```
void main (int argc, char** argv) {
    glutInit (&argc, argv);
    glutInitDisplay~~Window~~Mode (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 (3, GL_FLOAT, 0, normals);
    glColor3f (1, 1, 1);
    glutMainLoop();
}
```

Create menu with 3 entries: curves, colors, quit. Curves has sub menu with 4 entries Limacon, Cardioid, Three-Leaf, Spiral. Color menu has sub menu with all eight colors of RGB model. Write program.

```
#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; curve=1, red=0, green=0, blue=0;
void myinit () {
        glClearColor(1,1,1,1);
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(0,200,0,150); }

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 = 6.28318;
        const int a=175, b=60;
    float r, theta, dtheta = 1/(float)a;
    int x0=200, y0=250;
    screenPt curvePt[2];
    curve = curveNum;
    glColor3f(red, green, blue);
```

```
curvePt[0].x = x0;  curvePt[0].y = y0;
glClear(GL_COLOR_BUFFER_BIT);
switch(curveNum){
    case limacon: curvePt[0].x += a + b;  break;
    case cardioid: curvePt[0].x += a + a;  break;
    case threeLeaf: curvePt[0].x += a;  break;
    case spiral: break;
    default: break; }
theta = dtheta;
while(theta < twoPi){
    switch(curveNum){
    case limacon: r = a * cos(theta) + b; break;
    case cardioid: r = a * (1 + cos(theta)); break;
    case threeLeaf: r = a * cos(3 * theta); break;
    case spiral: r = (a/4.0) * theta; break;
    default: break; }
    curvePt[1].x = x0 + r * cos(theta);
    curvePt[1].y = y0 + r * sin(theta);
    lineSegment(curvePt[0], curvePt[1]);
    curvePt[0].x = curvePt[1].x;
    curvePt[0].y = curvePt[1].y;
    theta += dtheta;
  } }
void colorMenu(int id){
    switch(id){
        case 0: break;
        case 1: red = 0, green = 0, blue = 1;
                break;
        case 2: red = 0, green = 1, blue = 0;
                break;
```

```
case 3: red=0, green=1, blue=1;
        break;
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);
    /* printf ("enter integer value corresponding to following
    curve names \n 1- limacon\n 2- cardioid \n 3- threeleaf \n
    4 - spiral \n");
        scanf_s("%d", &curveNum);   */
        }

void myreshape (int nw, int nh) {
        glMatrixMode (GL_PROJECTION);
        glLoadIdentity ();
        gluOrtho2D(0, (double)nw, 0, (double)nh);
```

```
glClear (GL_COLOR_BUFFER_BIT); }

void main (int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize (w, h);
    glutInitWindowPosition (100,100);
    glutCreateWindow ("Drawing curves");
    int curveId = glutCreateMenu (drawCurve);
    glutAddMenuEntry ("Limacon", 1);
    glutAddMenuEntry ("Cardioid", 2);
    glutAddMenuEntry ("Threeleaf", 3);
    glutAddMenuEntry ("Spiral", 4);
    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);
    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();
}
```

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

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

using namespace std;
float f,g,r,xl[4], yc[4];
int flag=0;

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

void drawPixel(float x, float y){
        glBegin(GL_POINTS);
        glVertex2f(x,y);
        glEnd(); }

void display(){
        glClear(GL_COLOR_BUFFER_BIT);
        int i;
        double t;
        glColor3f(0,0,0);
        glBegin(GL_POINTS);
        for(t=0; t<1; t=t+0.005){
```

```
double xt = pow(1-t,3) * x1[0] + 3*t*pow(1-t,2)*x1[1]
  + 3* pow(t,2) *(1-t)*x1[2] + pow(t,3) *x1[3];

double yt= pow(1-t,3) * yc[0] + 3*t* pow(1-t,2) * yc[1] +
3* pow(t,2) * (1-t) * yc[2] + pow(t,3) * yc[3];

        glVertex 2f (xt, yt);
    }
  glColor3f(1, 1, 0);
  for(i=0; i<4 ; i++) {
          glVertex2f( x1[i], yc[i] );
          glEnd();
          glFlush(); }
                    }

void mymouse (int btn, int state, int x, int y) {
  if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN
        && flag <4) {
      x1 [ flag ] = x;
      yc [ flag ] = 500-y;
      cout << "x :" << x << " y: " << 500-y;
      glPointSize(3);
      glColor3f(1,1,0);
      glBegin(GL_POINTS);
      glVertex2i(x, 500-y);
      glEnd();
      glFlush();
      flag++; }
```

```
if(flag >=4 && btn == GLUT_LEFT_BUTTON){
        glColor3f (0,0,1);
        display();
        flag = 0;
    }
}

int main (int argc, char** argv){
            glutInit (&argc, argv);
/* USE KEYBOARD
   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];
//END KEYBOARD
   */

   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500,500);
   glutInitWindowPosition(0,0);
   glutCreateWindow ("BZ");
   glutDisplayFunc (display);
   glutMouseFunc (mymouse); //include for mouse
   myInit();
   glutMainLoop();
}
```

Certificate

Name: Sanjay S Hegde Class: 7C

Roll No: 1RV17CS139 Exam No:

Institution RV College of Engineering

This is certified to be the bonafide work of the student in the CG *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

S. No.	Name of the Experiment	Page No.	Date of Experiment	Date of Submission	Remarks
1.	Generate line using Bresenham's Line drawing technique.	3			
2.	Generate circle, ellipse with Bresenham's drawing techniques	7			
3.	3D Sierpinski gasket with recursive subdividing tetrahedron	15			
4.	Scan line area filling algorithm	19			
5.	Rotate and Reflect a house with OpenGL transformations	23			
6.	Line Clipping with Cohen-Sutherland algorithm	29			
7.	Line clipping with Liang Barsky algorithm	35			
8.	Polygon clipping with Cohen Hodgeman algorithm.	41			

Index

S. No.	Name of the Experiment	Page No.	Date of Experiment	Date of Submission	Remarks
9.	Drawing car with display lists and movement	47			
10.	Spinning color cube with OpenGL transformations	53			
11.	Menu with options to draw curves in different colors.	59			
12.	Constructing a Bezier curve	69			