R.V. COLLEGE OF ENGINEERING

(Autonomous Institution affiliated to VTU, Belagavi)

BENGALURU - 560 059.

Laboratory Certificate

This is to certify that RUJULA SINGHAL *has satisfactorily completed the course of Experiments in* COMPUTER GRAPHICS *Practical ..... Prescribed by the Department of* CSE *for the* 7th *Semester of BE Graduate Programme during the year 20 -20*

Signature of Head of the Department
Date :

Signature of the Faculty in-charge

50

Name of the Candidate .....

USN .....

Name o

Sl.No.

```
#include <iostream>
#include <GL/GLUT.h>
#include <time.h>
using namespace std;
int x1, x2, yc1, y2;
int flag = 0;
void draw_pixel(int x, int y)
{
        glColor3f(1, 0, 0);
        glBegin(GL_POINTS);
        glVertex2i(x, y);
        glEnd();
        glFlush();
}
void draw_line()
{
        int dx, dy, i, e;
        int incx, incy, inc1, inc2;
        int x, y;
        dx = x2 - x1;
        dy = y2 - yc1;
        if (dx < 0)dx = -dx;
        if (dy < 0)dy = -dy;
        incx = 1;
        if (x2 < x1)
                incx = -1;
        incy = 1;
        if (y2 < yc1)
                incy = -1;
        x = x1;
        y = yc1;
        if (dx > dy)
        {
                draw_pixel(x, y);
                e = 2 * dy - dx;
                inc1 = 2 * (dy - dx);
                inc2 = 2 * dy;
                for (i = 0; i < dx; i++)
                {
                        if (e > 0)
                        {
                                y += incy;
                                e += inc1;
                        }
                        else
                                e += inc2;
                        x += incx;
                        draw_pixel(x, y);
                }
        }
        else
        {
                draw_pixel(x, y);
                e = 2 * dx - dy;
                inc1 = 2 * (dx - dy);
                inc2 = 2 * dx;
                for (i = 0; i < dy; i++)
                {
                        if (e > 0)
                        {
                                x += incx;
                                e += inc1;
                        }
                        else
                                e += inc2;
                        y += incy;
```

Name of Experiment ..... Bresenham's line drawing ..... Date .....

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

write a program to generate a line using Bresenham's line drawing technique.

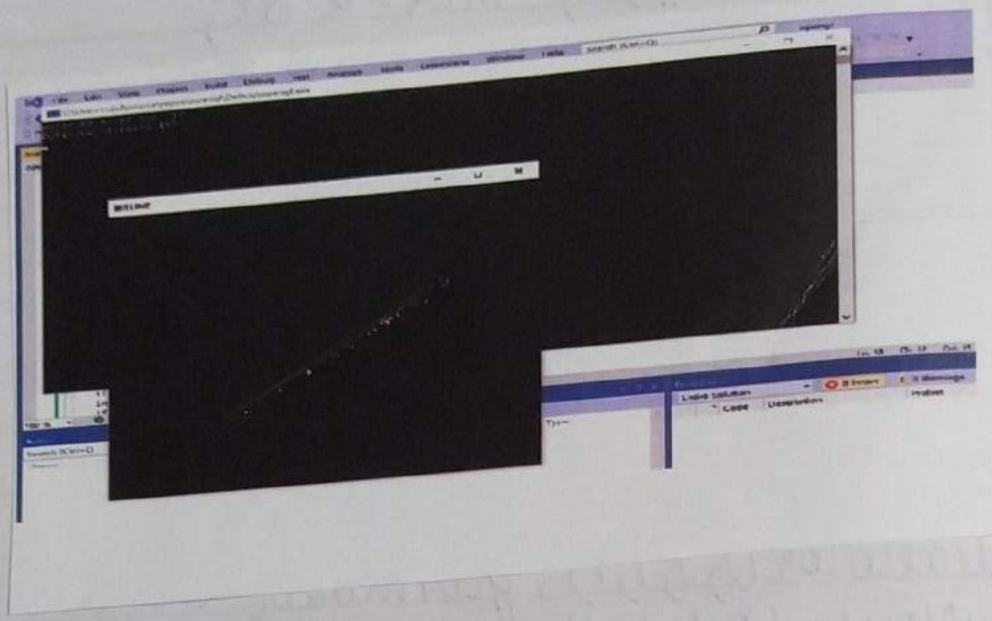
```
#include <iostream.h>
#include <GL/GLUT.h>
#include <time.h>
using namespace std;
int x1, x2, yc1, y2
void draw_pixel (int x, int y)
{   glColor3f(1,0,0)
    glBegin (GLPOINTS)
    glVertex2i (x,y)
    glEnd()
    glFlush() }
void DrawLine ()
{ int dx, dy, i, e, incx, incy, inc1, inc2, x, y;
  dx = x2-x1; dy = y2-yc1;
  if(dy<0) dy = -dy; if(dx<0) dx = -dx; incx = 1
  if (x2<x1) incx = -1; incy = 1
  if(y2<y1) incy = -1;
  x = x1; y = yc1;
  if(dx>dy) { draw_pixel(x,y)
  e = 2*dx-dy; inc1 = 2*(dx-dy); 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);
  }
}
```

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

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

Experiment No..... Page No.....

```
else {
   draw_pixel (x,y); c = 2*dx - dy;
   incl = 2*(dx - dy) ; inc2 = 2*dx;
   for( i=0; i<dy; i++) {
      if(c>0) { x+=incx ; c+=incl; }
      else  c+=inc2;
      y+=incy;
      draw_pixel (x, y); }
    { glFlush(); }
void myinit ()
{ glClear(GL_COLOUR_BUFFER_BIT);
   glClearColor(1,1,1,1); glOrtho2D(-250,250,-250,250); }
void MyMouse (int button, int state, int x, int y)
{ switch(button) {
   case GLUT_LEFT_BUTTON:
     { if (state==GLUT_DOWN)
       { if(flag==0) {
          printf("Defining x1, y1 ");
          x1=x-250; y1= 250-y; flag++;
          cout<< x1<<" " << y1; }
        else { printf("Defining x2, y2")
         x2=x-250; y2= 250-y; flag=0;
         cout<< x2 <<" " << y2<<"\n"; draw_line(); }
        } break;
    }
}
```



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

Experiment No..... Page No.....

```
void display ()
{}

int main (int ac, char *av[])
{  glutInit (&ac, av);
   glutInitDisplay (GLUT_SINGLE | GLUT_RGB)
   glutInitWindowSize (500, 500)
   glutInitWindowPosition (100, 200);
   glutCreateWindow ("LINE");
   myinit ();
   glutMouseFunc (MyMouse);
   glutDisplayFunc (display);
   glutMainLoop ();
}
```

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

Name of Experiment: Bresenham's Circle & Midpoint Ellipse Date:

Experiment No: D2 Page No:

write a program to generate Circle & Ellipse.

```
#include <gl/glut.h>
#include <stdio.h>
#include <math.h>
int xc,yc,r; rx, ry, xce, yce;
int 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+ry, yc+rx); glVertex2i(xc-ry, yc+rx)
  glVertex2i(xc+ry, yc-rx); glVertex2i(xc-ry, yc-rx)
  glEnd() }
void bres_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(); }
int p1_x,p2_x,p1_y,p2_y; points_done=0;
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();
}
```

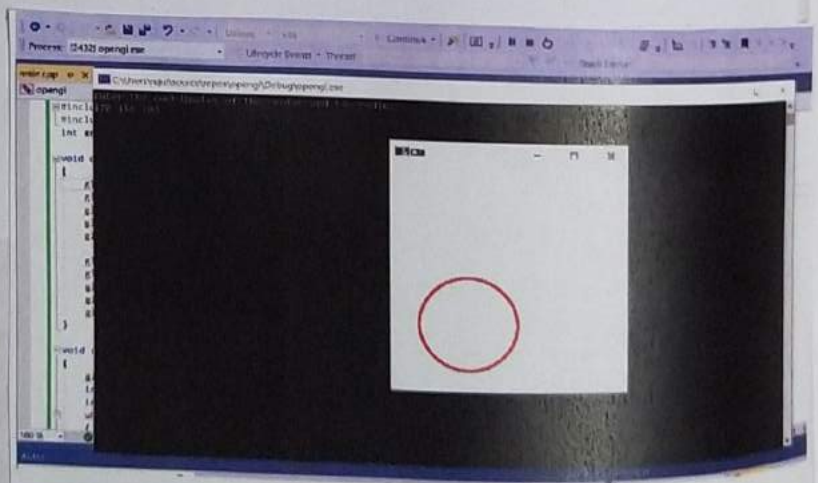
```
        glVertex2i(x + xce, -y + yce);
        glVertex2i(-x + xce, -y + yce);
        glEnd();
}
void midptellipse()
{
        glClear(GL_COLOR_BUFFER_BIT);
        float dx, dy, d1, d2, x, y;
        x = 0;
        y = ry;
        d1 = (ry * ry) - (rx * rx * ry) +
                (0.25 * rx * rx);
        dx = 2 * ry * ry * x;
        dy = 2 * rx * rx * y;

        // For region 1
        while (dx < dy)
        {
                draw_ellipse(xce, yce, x, y);
                if (d1 < 0)
                {
                        x++;
                        dx = dx + (2 * ry * ry);
                        d1 = d1 + dx + (ry * ry);
                }
                else
                {
                        x++;
                        y--;
                        dx = dx + (2 * ry * ry);
                        dy = dy - (2 * rx * rx);
                        d1 = d1 + dx - dy + (ry * ry);
                }
        }

        // Decision parameter of region 2
        d2 = ((ry * ry) * ((x + 0.5) * (x + 0.5))) +
                ((rx * rx) * ((y - 1) * (y - 1))) -
                (rx * rx * ry * ry);

        // Plotting points of region 2
        while (y >= 0)
        {

                // Print points based on 4-way symmetry
                draw_ellipse(xce, yce, x, y);

                // Checking and updating parameter
                // value based on algorithm
                if (d2 > 0)
                {
                        y--;
                        dy = dy - (2 * rx * rx);
                        d2 = d2 + (rx * rx) - dy;
                }
                else
                {
                        y--;
                        x++;
                        dx = dx + (2 * ry * ry);
                        dy = dy - (2 * rx * rx);
                        d2 = d2 + dx - dy + (rx * rx);
                }
        }
        glFlush();
}
```

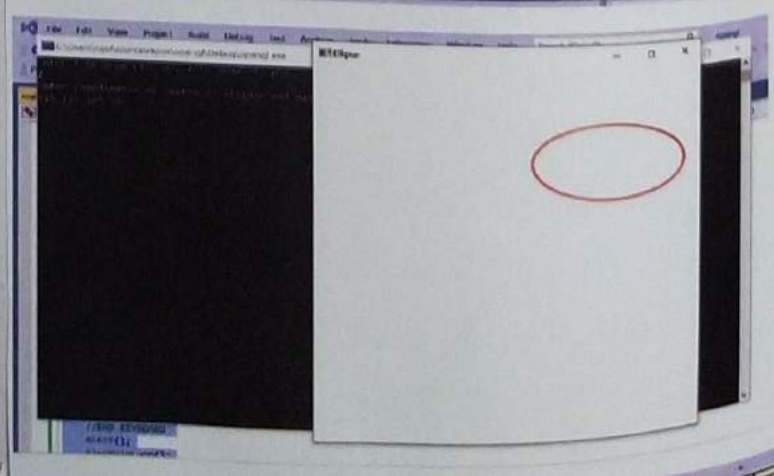
Name of Experiment ..... Date .....

Experiment No..... Page No.....

```
void mid_ellipse()
{ glClear(GL_COLOUR_BUFFER_BIT)
  float dx, dy, d1, d2, xy; x=0; y=ry;
  d1=(ry*ry)-(rx*rx*rx)+(0.25*rx*rx);
  dx=2*ry*ry*x; dy=2*rx*rx*y;
  while(dx<dy)
  { draw_ellipse(xc, yc, x, y);
    if(d1<0) { x++; dx=dx+(2*ry*ry);
      d1=d1+dx+(ry*ry); }
    else { x++; y--; dx=dx+(2*ry*ry);
      dy=dy-(2*rx*rx); d1=d1+dx-dy+(ry*ry); }
  }
  d2=((ry*ry)*((x+0.5)*(x+0.5)))+((rx*rx)*((y-1)*(y-1)))-
    (rx*rx*ry*ry);
  while(y>=0) {
   draw_ellipse(xc, yc, x, y);
   if(d2>0) { y--; dy=dy-(2*rx*rx); d2=d2+(rx*rx)-dy; }
   else { y--; x++; dx=dx+(2*ry*ry)
     dy=dy-(2*rx*rx); d2=d2+dx-dy+(rx*rx); }
   } glFlush();
}
void main(int argc, char *argv[])
{ glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(0,0);
  printf("Enter 1 for circle 2 for Ellipse");
  int ch; scanf("%d", &ch);
```

```
int p1e_x, p2e_x, p1e_y, p2e_y, p3e_x, p3e_y;
int point1e_done = 0;
void minit()
{
        glClearColor(1, 1, 1, 1);
        glColor3f(1.0, 0.0, 0.0);
        glPointSize(3.0);
        gluOrtho2D(-250, 250, -250, 250);
}

void main(int argc, char* argv[])
{

        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(0, 0);
        printf("Enter 1 to draw circle , 2 to draw ellipse\n");
        int ch;
        scanf_s("%d", &ch);
        switch (ch) {
        case 1:
                printf("Enter coordinates of centre of circle and radius\n");
                scanf_s("%d%d%d", &xc, &yc, &r);
                glutCreateWindow("Circle");
                glutDisplayFunc(circlebres);
                break;
        case 2:
                printf("Enter coordinates of centre of ellipse and major and minor radius\n");
                scanf_s("%d%d%d%d", &xce, &yce, &rx, &ry);
                glutCreateWindow("Ellipse");
                glutDisplayFunc(midptellipse);
                break;
        }
        minit();
        glutMainLoop();
}
```





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

Experiment No..... Page No.....

```
switch(ch){
case1 : printf("Enter coordinates of center of circle & radius");
        scanf("%d %d %d", &xc, &yc, &r)
        glutCreateWindow("Circle");
        glutDisplayFunc(bres_circle); break;
case 2: printf("Enter coordinates of center of Ellipse and
        major and minor radius");
        scanf("%d %d %d %d", &xce, &yce, &rx, &ry);
        glutCreateWindow("Ellipse");
        glutDisplayFunc(midEllipse); break;
}
init()
glutMainloop();
}
```

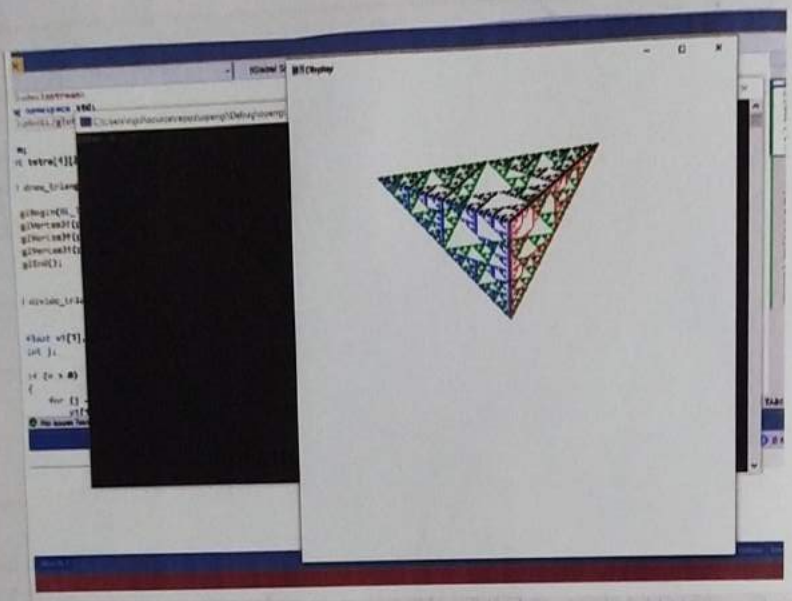
```
#include<iostream>
using namespace std;
#include<GL/glut.h>
int m;
float tetra[4][3] = { { 0, 200, 400 }, { 0, 0, -350 }, { 200, 350, 300 }, { -300, 300, 200 } };
void draw_triangle(float p1[], float p2[], float p3[])
{
        glBegin(GL_TRIANGLES);
        glVertex3f(p1[0], p1[1], p1[2]);
        glVertex3f(p2[0], p2[1], p2[2]);
        glVertex3f(p3[0], p3[1], p3[2]);
        glEnd();
}
void divide_triangle(float a[], float b[], float c[], int m)
{
        float v1[3], v2[3], v3[3]; 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] = (c[j] + b[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 tetrahedron()
{
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(1.0, 0, 0);
        divide_triangle(tetra[0], tetra[1], tetra[2], m);
        glColor3f(0, 1.0, 0);
        divide_triangle(tetra[3], tetra[2], tetra[1], m);
        glColor3f(0, 0, 1.0);
        divide_triangle(tetra[0], tetra[3], tetra[1], m);
        glColor3f(0, 0, 0);
        divide_triangle(tetra[0], tetra[2], tetra[3], m);
        glFlush();
}
void myinit()
{
        glClearColor(1, 1, 1.0, 1.0);
        glColor3f(1.0, 0, 0);
        glPointSize(5.0);
        glOrtho(-500, 500, -500, 500, -500, 500);
}

void main(int argcp, char** argv)
{
        cout << "Enter m : ";
        cin >> m;
        glutInit(&argcp, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_DEPTH);
        glutInitWindowSize(300, 300);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("Display");
        glutDisplayFunc(tetrahedron);
        glEnable(GL_DEPTH_TEST);
        myinit();
        glutMainLoop();
}
```

Name of Experiment: Sierpenski Gasket Date:

Experiment No: 03 Page No:

Write a program that recursively subdivides a tetrahedron to form 3D Sierpenski Gasket.

```
#include<GL/glut.h>
#include<iostream.h>
using namespace std;
int m;
float tetra[4][3]={{0,200,400},{0,0,-350},{200,350,300},{-300,300,200}};
void draw_triangle(float p1[],float p2[],float p3[])
{ glBegin(GL_TRIANGLES)
  glVertex3f(p1[0],p1[1],p1[2]);
  glVertex3f(p2[0],p2[1],p2[2]);
  glVertex3f(p3[0],p3[1],p3[2]);
  glEnd() }
void divide_triangle(float a[],float b[],float c[],int m)
{ float v1[3],v2[3],v3[3]; 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]=(c[j]+b[j])/2;
    divide_triangle(a,v1,v3,m-1);
    divide_triangle(c,v2,v3,m-1);
    divide_triangle(b,v3,v1,m-1);
  }else draw_triangle(a,b,c);
}
void tetrahedron()
{ glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT)
  glColor3f(1.0,0,0);
  divide_triangle(tetra[0],tetra[1],tetra[2],m);
```



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

Experiment No..... Page No.....

```
glColor3f(0, 1.0, 0)
divide_triangles(tetra[3], tetra[2], tetra[1], m)
glColor3f(0, 0, 1.0)
divide_triangles(tetra[0], tetra[3], tetra[1], m)
glColor3f(1.0, 1.0, 0)
divide_triangles(tetra[0], tetra[2], tetra[3], m);
glFlush() }
void minit() {
  glClearColor(1, 1, 1.0, 1.0); glColor3f(1.0, 0, 0)
  glPointSize(5.0); glOrtho(-500, 500, -500, 500, -500, 500); }
void main(int argc, char **argv)
{  cout << "Enter m:";
   cin >> m;
   glutInit(&argc, argv)
   glutInitDisplayMode(GLUT_SINGLE | GLUT_DEPTH)
   glutInitWindowSize(300, 300)
   glutInitWindowPosition(100, 100);
   glutCreateWindow("Display");
   glutDisplayFunc(Tetrahedron);
   glEnable(GL_DEPTH_TEST);
   minit()
   glutMainLoop();
}
```

```
#include<stdlib.h>
#include<gl/glut.h>
#include<algorithm>
#include<iostream>
#include<windows.h>
using namespace std;
float x[100], y[100];
int n, m;
int wx = 500, wy = 500;
static float intx[10] = { 0 };
void draw_line(float x1, float y1, float x2, float y2) {
        Sleep(100);
        glColor3f(1, 0, 0);
        glBegin(GL_LINES);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
        glEnd();
        glFlush();
}
void edgeDetect(float x1, float y1, float x2, float y2, int scanline) {
        float temp;
        if (y2 < y1) {
                temp = x1; x1 = x2; x2 = temp;
                temp = y1; y1 = y2; y2 = temp;
        }
        if (scanline > y1 && scanline < y2)
                intx[m++] = x1 + (scanline - y1) * (x2 - x1) / (y2 - y1);
}
void 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);
}
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_s("%d", &n);
        printf("Enter coordinates of endpoints: \n");
        for (int i = 0; i < n; i++)
        {
                printf("X-coord Y-coord: \n");
                scanf_s("%f %f", &x[i], &y[i]);
        }
```

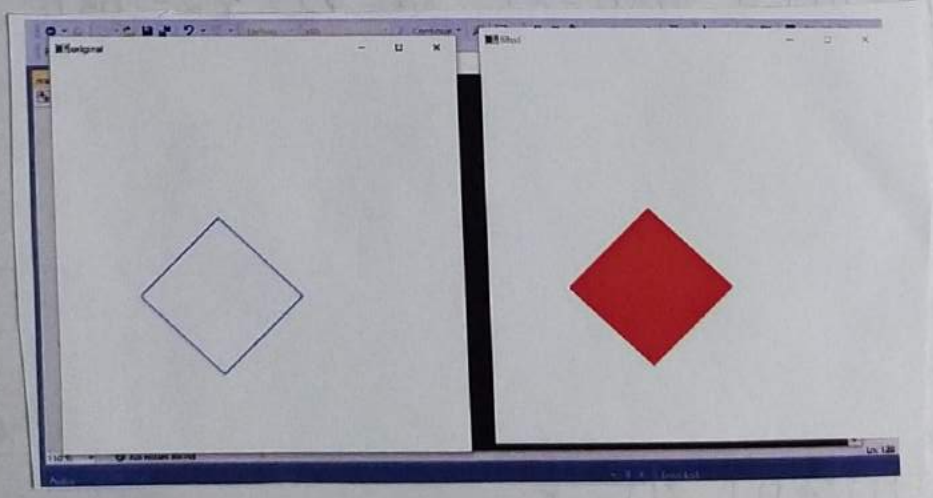
Name of Experiment 04 SCAN FILL Date

Experiment No. 04 Page No.

write a program to fill a polygon using scan-a-fill algo.

```
#include<gl/glut.h>
#include<math.h><iostream.h>
#include<stdlib.h>
#include<windows.h>
#include<algorithm>
using namespace std;
float x[100], y[100]; int n, m; int wx=500, wy=500;
static float int x[10] = {0};
void edgeDetect(float x1, float y1, float x2, float y2, int scanline)
{ Sleep(10);
  glColor3f(1,0,0); glBegin(GL_LINES); glVertex2f(x1,y1);
  glVertex2f(x2,y2); glEnd();
}
Void EdgeDetect(float x1, float y1, float x2, float y2, int scanline)
{ float temp;
  if(y2<y1) { temp=x1; x2=y2; 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)
  drawline(intx[i], s1, intx[i+1], s1); }}
```

```
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0, 0);
    glutCreateWindow("scanline");
    glutDisplayFunc(display_filled_polygon);
    myInit();
    glutMainLoop();

}
```



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

Experiment No..... Page No.....

```
void display-filled-polygon( ) {
glClear(GL_COLOR_BUFFER_BIT); glLineWidth(2);
glBegin(GL_LINE_LOOP);
for(int i=0; i<n; i++) glVertex2f(x[i], y[i]);
glEnd(); scanfill(x,y);
}
void myinit() { glClearColor(1,1,1,1); glColor3f(0,0,1);
glPointSize(1)
glOrtho2D(0, wx, 0, wy);
}
void main(int a, char* v[])
{
    glutInit(&a, v);
    printf("Enter no of sides : \n"); scanf-s("%d", &n);
    for(int i=0; i<n; i++)
    { printf("x-coord Y coord : \n");
      scanf-s("%f %f", &x[i], &y[i]); }
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB)
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Scan line");
    glutDisplayFunc(display-filled-polygon);
    myinit();
    glutMainLoop();
}
```

```
#include<gl/glut.h>
#include <math.h>
#include<stdio.h>
float house[11][2] = { { 100,200 },{ 200,250 },{ 300,200 },{ 100,200 },{ 100,100 },{ 175,100 },{ 175,150 },{ 225,150 },{ 225,100 },{ 300,10
300,200 } };
int angle;
float m, c, theta;
void display()
{
        glClearColor(1, 1, 1, 0);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-450, 450, -450, 450);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glColor3f(1, 0, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glFlush();
        glPushMatrix();
        glTranslatef(100, 100, 0);
        glRotatef(angle, 0, 0, 1);
        glTranslatef(-100, -100, 0);
        glColor3f(1, 1, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glPopMatrix();
        glFlush();
}
void display2()
{
        glClearColor(1, 1, 1, 0);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-450, 450, -450, 450);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        glColor3f(1, 0, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glFlush();
        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();
        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);
```

Name of Experiment House Rotation & Reflection Date

Experiment No. 05 Page No.

Write a program to create house like figure and
i) Reflect it about given fixed point using OPENGL.
ii) Reflect it about an axis $y = mx + c$.

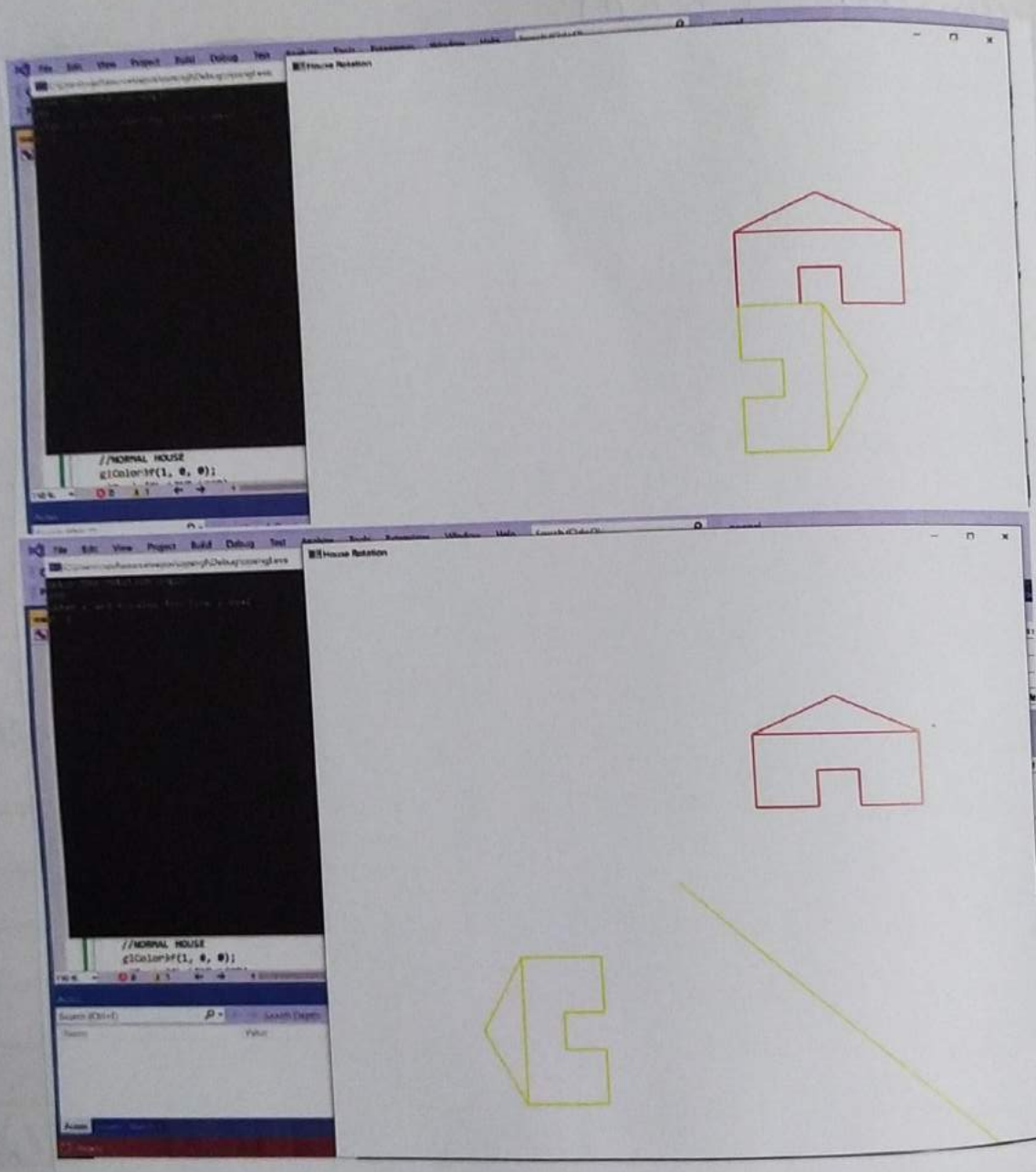
```
#include <gl/glut.h>
#include <math.h>
#include <stdio.h>
float house[11][2] = {{100,200}, {200,250}, {300,200},
{100,200}, {100,100}, {175,100}, {175,150}, {225,150}, {300,100},
{300,200}};
int angle;
float m, c, theta;
void display()
{   glClearColor(1,1,1,0);
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glOrtho2D(-450, 450, -450, 450);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glColor3f(1,0,0);
    glBegin(GL_LINE_LOOP)
    for(i=0; i<11; i++)
        glVertex2fv(house[i]);
    glEnd(); glFlush(); glPushMatrix();
    glTranslatef(100,100,0) ; glRotatef(angle, 0,0,1);
    glTranslatef(-100,-100,0) ; glColor3f(1,1,0)
    glBegin(GL_LINE_LOOP); for(i=0; i<11; i++)
    glVertex2fv(house[i]); glEnd(); glPopMatrix glFlush();
}
```

```
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glPopMatrix();
        glFlush();
}
void myInit() {
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(1.0, 0.0, 0.0);
        glLineWidth(2.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-450, 450, -450, 450);
}
void mouse(int btn, int state, int x, int y) {
        if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
                display();
        }
        else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
                display2();
        }
}
void main(int argc, char** argv)
{
        printf("Enter the rotation angle\n");
        scanf_s("%d", &angle);
        printf_s("Enter c and m value for line y=mx+c\n");
        scanf_s("%f %f", &c, &m);
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(900, 900);
        glutInitWindowPosition(100, 100);
        glutCreateWindow("House Rotation");
        glutDisplayFunc(display);
        glutMouseFunc(mouse);
        myInit();
        glutMainLoop();
}
```

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

Experiment No. ..... Page No. .....

```
void display2()
{ glClearColor(1,1,1,0)
  glClear(GL_COLOR_BUFFER_BIT | GL_BUFFER_BIT);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  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();
  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();
  glPushMatrix()
  glTranslatef(0, c, 0);
  theta = atan(m);
  theta = theta* 180/3.34;
  glRotatef(-theta; 0, 0, 1);
  glScalef(1, -1, 1); 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();
}
```


House Rotation
//NORMAL HOUSE
glColor3f(1, 0, 0);
House Rotation
//NORMAL HOUSE
glColor3f(1, 0, 0);

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

Experiment No. ..... Page No. .....

```
myinit() {
  glClearColor(1.0, 1.0, 1.0, 1.0);
  glColor3f(1.0, 0.0, 0.0);
  glLineWidth(2.0);
  glMatrixMode(GL_PROJECTION_MODE
  glLoadIdentity();
  gluOrtho2D(-450, 450, -450, 450); }
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");
  scanf_s("%d", &angle);
  printf_s("Enter c and m value for reflection");
  scanf_s("%f%f", &m, &c);
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
  glutInitWindowSize(900, 900);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Rotation");
  glutDisplayFunc(display);
  glutMouseFunc(mouse);
  myinit();
  glutMainLoop();
}
```

```
#include<stdio.h>
#include<stdlib.h>
#include<gl/glut.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)
                        done = true;
                else
                {
                        double x, y;
                        outcodeout = outcode0 ? outcode0 : outcode1;
                        if (outcodeout & TOP)
                        {
                                x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
                                y = ymax;
                        }
                        else if (outcodeout & BOTTOM)
                        {
                                x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
                                y = ymin;
                        }
                        else if (outcodeout & RIGHT)
                        {
                                y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
                                x = xmax;
```

Name of Experiment ..... Cohen-Sutherland Line-clipping. Date .....

Experiment No. 06 Page No. .....

write a program to implement Cohen-Sutherland line clipping algorithm

```
#include<stdio.h>
#include<stdlib.h>
#include<gl/glut.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 TOP = 1
const int LEFT = 8;   const int BOTTOM = 2;  int n;
struct lineSegment { int x1, y1, x2, y2; };
struct lineSegment ls[10];
outcode computecode (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 = computecode(x0, y0)
    outcode1 = computecode(x1, y1).
  do { if (!(outcode0 | outcode1)) { accept = true; done = true; }
    else if (outcode0 & outcode1) done = true;
    else { double x, y;
    outcodeout = outcode0 ? outcode0 : outcode1;
```

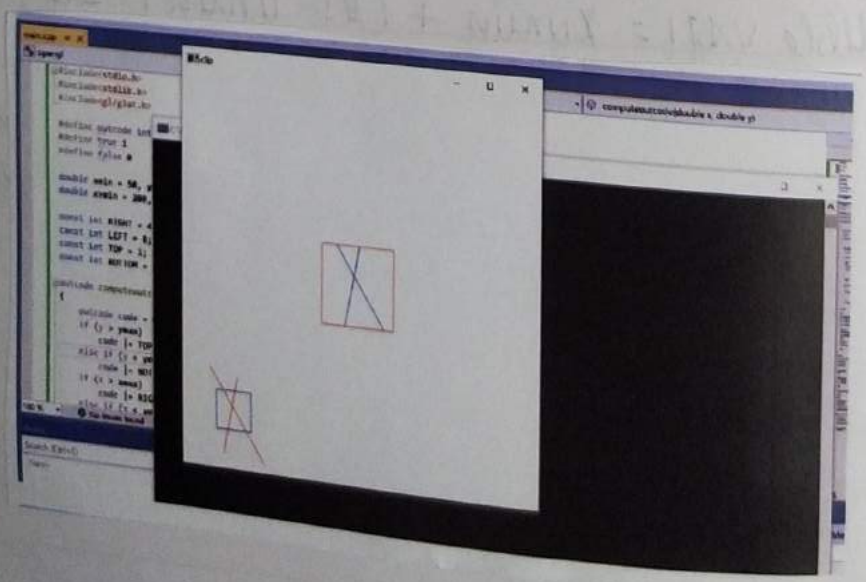
```
                }
                else
                {
                        y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
                        x = xmin;
                }

                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 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);
        glBegin(GL_LINE_LOOP);
        glVertex2f(xvmin, yvmin);
        glVertex2f(xvmax, yvmin);
        glVertex2f(xvmax, yvmax);
        glVertex2f(xvmin, yvmax);
        glEnd();

        glColor3f(0, 0, 1);
        glBegin(GL_LINES);
        glVertex2d(vx0, vy0);
        glVertex2d(vx1, vy1);
        glEnd();
    }
}

void display()
{
    glClear(GL_COLOR_BUFFER_BIT);

    glColor3f(0, 0, 1);
    glBegin(GL_LINE_LOOP);
    glVertex2f(xmin, ymin);
    glVertex2f(xmax, ymin);
    glVertex2f(xmax, ymax);
    glVertex2f(xmin, ymax);
    glEnd();
    for (int i = 0; i < n; i++)
    {
        glBegin(GL_LINES);
        glVertex2d(ls[i].x1, ls[i].y1);
        glVertex2d(ls[i].x2, ls[i].y2);
        glEnd();
    }
```

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

Experiment No..... Page No.....

```
if(outcodeout & TOP)
{ x=x0 +(x1-x0) * (ymax - y0) /(y1 -y0); y =ymax ; }
else if (outcodeout & BOTTOM)
{ x=x0+ (x1-x0) * (ymin - y0) / y1 - y0); y=ymin; }
else if (outcodeout & RIGHT)
{ y=y0+ (y1-y0) * (xmax-x0)/(x1-x0); x=xmax; }
else { y=y0+(y1-y0) * (xmin-x0) /(x1-x0) ; x= xmin; }
if (outcodeout == outcode0) { x0= x; y0= y; outcode0=
    computecode(x0,y0) ; }
else { x1=x; y1=y; outcode1 =computecodecode(x1,y1); }
} while(!done);
if(accept)
{ double sx= (xvmax -xvmin)/(xmax -xmin);
  double sy = (yvmax- yvmin)/ (ymax -ymin)
  double vx0 = xvmin + (x0 - xmin) * sx;
  double vy0 = yvmin + (y0 - ymin) * sy;
  double vx1 = xvmin + (x1 - xmin) * sx;
  double vy1 = xymin + (y1 - ymin) * sy;
  glColor3f(1,0,0) glBegin(GL_LINE_LOOP);
  glVertex2f(xvmin, yvmin) glVertex2f (xvmax, yvmin)
  glVertex2f (xvmax, ymax); glVertex2f (xvmin, yvmax);
  glEnd();
  glColor3f(0,0,1);
  glBegin(GL_LINES);
  glVertex2d (vx0, vy0);
  glVertex2d (vx1, vy1);
  glEnd();
}
}
```

```
        for (int i = 0; i < n; i++)
                cohensuther(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);
}

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



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

Experiment No..... Page No.....

```
void display()
{ glClear(GL_COLOR_BUFFER_BIT)
  glColor3f(0,0,1)
  glBegin(GL_LINE_LOOP)
  glVertex2f(xmin,ymin);
  glVertex2f(xmax,ymin); glVertex(xmax, ymax);
  glVertex(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++)
     colorsutherlines(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);
}

void main(int c, char** v)
{ printf("enter window & viewport coordinates");
  scanf("%f%f%f%f%f%f%f", &xmin,&ymin, &xmax,&ymax);
  printf("Enter no of lines"); scanf("%d", &n);
  for(i=0; i<n; i++)
    { printf("Enter line coordinates"); scanf("%d%d%d%d",&ls[i].x1,
      &ls[i].y1), &ls[i].y1, &ls[i].y2); }
  glutInit(&c,v); glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500,500); glutInitWindowPosition(0,0);
glutCreateWindow("Clip"); myinit(); glutDisplayFunc(display);
  glutMainLoop(); }
```

```
#include <stdio.h>
#include <GL/glut.h>
#include <iostream>
using namespace std;
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);
        glEnd();
        if (cliptest(-dx, x0 - xmin, &u1, &u2))  // inside test wrt left edge
                if (cliptest(dx, xmax - x0, &u1, &u2)) // inside test wrt right edge
                        if (cliptest(-dy, y0 - ymin, &u1, &u2)) // inside test wrt bottom edge
                                if (cliptest(dy, ymax - y0, &u1, &u2)) // inside test wrt top edge
                                {
                                        if (u2 < 1.0)
                                        {
                                                x1 = x0 + u2 * dx;
                                                y1 = y0 + u2 * dy;
                                        }
                                        if (u1 > 0.0)
                                        {
                                                x0 = x0 + u1 * dx;
                                                y0 = y0 + u1 * dy;
                                        }
                                        // Window to viewport mappings
                                        double sx = (xvmax - xvmin) / (xmax - xmin); // Scale parameters
                                        double sy = (yvmax - yvmin) / (ymax - ymin);
```

Name of Experiment: Liang Barsky Line-clipping Date:

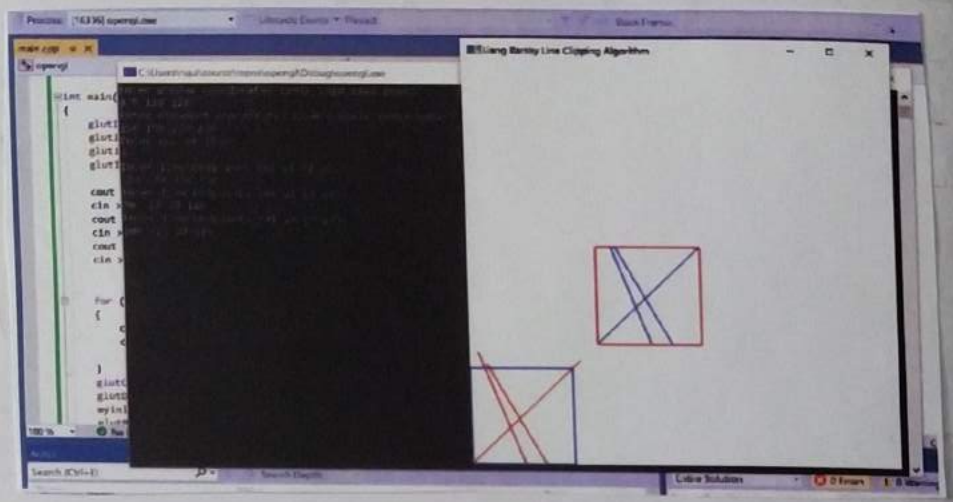
Experiment No.: Program-7 Page No.:

Write a program to implement Liang Barsky Algorithm.

```
#include<stdio.h>
#include<GL/glut.h>
#include<iostream.h>
using namespace std;
double xvmin, yvmin, xvmax, yvmax, xmin, ymin, xmax, ymax; 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<0.0) { if(r>*u1) *u1=r;
        if(r>*u2) return(false); }
  else if(p>0.0) { if(r<*u2) *u2=r; if(r<*u1) return false; }
       else if(p==0) { if(q<0.0) return false; }
  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;
  glColor3f(1,0,0)
  glBegin(GL_LINE_LOOP);
  glVertex2f(xvmin, yvmin); glVertex2f(xvmax, yvmin);
  glVertex2f(xvmax, yvmax); glVertex2f(xvmin, yvmax); glEnd();
if(cliptest(-dx, x0-xmin, &u1, &u2))
  if(cliptest(dx, xmax-x0, &u1, &u2))
    if(cliptest(-dy, y0-ymin, &u1, &u2))
      if(cliptest(dy, ymax-y0, &u1, &u2))
        { if(u2<1.0) { x1=x0+u2*dx; y1=y0+u2*dy; }
          if(u1>0.0) { x0=x0+u1*dx; y0=y0+u1*dy; }
```

```
                                        double vx0 = xvmin + (x0 - xmin) * sx;
                                        double vy0 = yvmin + (y0 - ymin) * sy;
                                        double vx1 = xvmin + (x1 - xmin) * sx;
                                        double vy1 = yvmin + (y1 - ymin) * sy;
                                        glColor3f(0.0, 0.0, 1.0); // draw blue colored clipped line
                                        glBegin(GL_LINES);
                                        glVertex2d(vx0, vy0);
                                        glVertex2d(vx1, vy1);
                                        glEnd();
                                }
        }
        void display()
        {
                glClear(GL_COLOR_BUFFER_BIT);
                glColor3f(1.0, 0.0, 0.0);
                for (int i = 0; i < n; i++)
                {
                        glBegin(GL_LINES);
                        glVertex2d(ls[i].x1, ls[i].y1);
                        glVertex2d(ls[i].x2, ls[i].y2);
                        glEnd();
                }
                //draw a blue colored window
                glColor3f(0.0, 0.0, 1.0);
                glBegin(GL_LINE_LOOP);
                glVertex2f(xmin, ymin);
                glVertex2f(xmax, ymin);
                glVertex2f(xmax, ymax);
                glVertex2f(xmin, ymax);
                glEnd();
                for (int i = 0; i < n; i++)
                        LiangBarskyLineClipAndDraw(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);
                glFlush();
        }
        void myinit()
        {
                glClearColor(1.0, 1.0, 1.0, 1.0);
                glColor3f(1.0, 0.0, 0.0);
                glLineWidth(2.0);
                glMatrixMode(GL_PROJECTION);
                glLoadIdentity();
                gluOrtho2D(0.0, 499.0, 0.0, 499.0);
        }
        int main(int argc, char** argv)
        {
                glutInit(&argc, argv);
                glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
                glutInitWindowSize(500, 500);
                glutInitWindowPosition(0, 0);
                cout << "Enter window coordinates (xmin ymin xmax ymax): \n";
                cin >> xmin >> ymin >> xmax >> ymax;
                cout << "Enter viewport coordinates (xvmin yvmin xvmax yvmax) : \n";
                cin >> xvmin >> yvmin >> xvmax >> yvmax;
                cout << "Enter no. of lines:\n";
                cin >> n;
                for (int i = 0; i < n; i++)
                {
                        cout << "Enter line endpoints (x1 y1 x2 y2):\n";
                        cin >> ls[i].x1 >> ls[i].y1 >> ls[i].x2 >> ls[i].y2;

                }
                glutCreateWindow("Liang Barsky Line Clipping Algorithm");
                glutDisplayFunc(display);
                myinit();
                glutMainLoop();
        }
```

```
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(0, 0, 1); glBegin(GL_LINES) glVertex2d(vx0, vy0);
glVertex2d(vx1, vy1); glEnd(); } }
void display()
{ glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 0.0, 0.0);
  for(int i=0; i<n; i++)
  { glBegin(GL_LINES); glVertex2d(ls[i].x1, ls[i].y1);
    glVertex2d(ls[i].x2, ls[i].y2); glEnd(); }
  glColor3d(0,0,1)
  glBegin(GL_LINE_LOOP)
  glVertex2f(xmin, ymin); glVertex(xmin, ymax); glVertex2f(xmax, ymax);
  glVertex(xmax, ymin); glEnd();
  for(int i=0; i<n; i++)
      LiangBarskyLineClipAndDraw(ls[i].x1, ls[i].y1, ls[i].x2, ls[i].y2);
  glFlush();
}
void myinit()
{ glClearColor(1.0, 1.0, 1.0, 1.0);
  glColor3f(1,0,0)
  glLineWidth(2.0);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0, 499, 0, 499);
}
```


Liang Barsky Line Clipping Algorithm

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

Experiment No..... Page No.....

```
int main (int argc, char **argv)
{ glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutInitWindowPosition (0, 0);
  cout<<" enter window coordinates ";
  cin >>xmin>>ymin >>xmax >>ymax;
  cout << "enter viewport coordinates";
  cin >> xmax >> ymax>>xvmin >> yvmin;
  cout << "enter no of lines";
  cin>>n;
  for(int i=0 ;i<n ; i++)
   { cout << enter line endpoints";
     cin>> ls[i].x1 >> ls[i].y1 >> ls[i].x2>>ls[i].y2 ; }
  glutCreateWindow ("LiangBarsky Line Clipping");
  glutDisplayFunc (display);
  myinit();
  glutMainLoop();
}
```

```
#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;
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;
}
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;
}
void clip(int poly_points[][2], int& poly_size,
        int x1, int y1, int x2, int y2)
{
        int new_points[MAX_POINTS][2], new_poly_size = 0;
        for (int i = 0; i < poly_size; i++)
        {
                int k = (i + 1) % poly_size;
                int ix = poly_points[i][0], iy = poly_points[i][1];
                int kx = poly_points[k][0], ky = poly_points[k][1];
                int i_pos = (x2 - x1) * (iy - y1) - (y2 - y1) * (ix - x1);
                int k_pos = (x2 - x1) * (ky - y1) - (y2 - y1) * (kx - x1);
                if (i_pos >= 0 && k_pos >= 0)
                {
                        //Only second point is added
                        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)
                {
                        // Point of intersection with edge
                        // and the second point is added
                        new_points[new_poly_size][0] = x_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_points[new_poly_size][1] = y_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_poly_size++;

                        new_points[new_poly_size][0] = kx;
                        new_points[new_poly_size][1] = ky;
                        new_poly_size++;
                }
                else if (i_pos >= 0 && k_pos < 0)
                {
                        //Only point of intersection with edge is added
                        new_points[new_poly_size][0] = x_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
                        new_points[new_poly_size][1] = y_intersect(x1,
                                y1, x2, y2, ix, iy, kx, ky);
```

Name of Experiment: Cohen-Hodgman Polygon clipping Date:

Experiment No: 08 Page No:

Write a program to implement Cohen-Hodgeman algorithm.

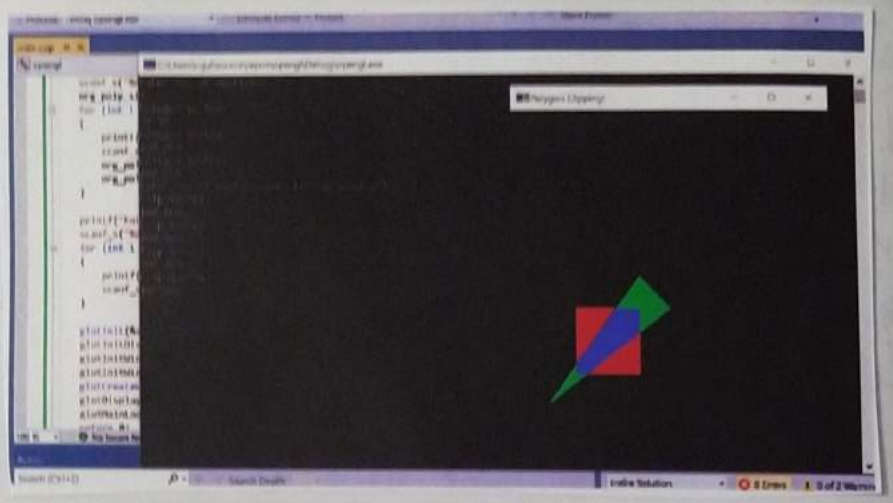
```
#include <iostream.h>
#include <GL/glut.h>
using namespace std;
int polysize, poly_points[20][2], org_poly_size, org_poly_points[20][2],
clipper_size, clipper_points[20][2];
const int max_points = 20;
void DrawPoly(int p[][2], int n)
{ glBegin(GL_POLYGON)
    for(int i=0; i<n; i++)
        glVertex2f(p[i][0], 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; }
int y_intersect(int x1, int y1, x2, y2, x3, y3, x4, 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; }
void clip(int poly_points[][2], int &poly_size, int x1, int y1, int x2, int y2)
{ int new_points[MAX_POINTS][2]; new_poly_size = 0;
  for(int i=0; i<poly_size; i++)
  { int k = (i+1) % poly_size;
    int ix = poly_points[i][0], iy = poly_points[i][1];
    int kx = poly_points[k][0], ky = poly_points[k][1];
    int i_pos = (x2-x1) * (iy-y1) - (y2-y1) * (ix-x1);
    int k_pos = (x2-x1) * (ky-y1) - (y2-y1) * (kx-x1);
```

```
                    new_poly_size++;
                }
                else{}
        }
        poly_size = new_poly_size;
        for (int i = 0; i < poly_size; i++)
        {
                poly_points[i][0] = new_points[i][0];
                poly_points[i][1] = new_points[i][1];
        }
}
void 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()
{
        init();
        glColor3f(1.0f, 0.0f, 0.0f);
        drawPoly(clipper_points, clipper_size);
        glColor3f(0.0f, 1.0f, 0.0f);
        drawPoly(org_poly_points, org_poly_size);
        for (int i = 0; i < clipper_size; i++)
        {
                int k = (i + 1) % clipper_size;
                clip(poly_points, poly_size, clipper_points[i][0],
                        clipper_points[i][1], clipper_points[k][0],
                        clipper_points[k][1]);
        }
        glColor3f(0.0f, 0.0f, 1.0f);
        drawPoly(poly_points, poly_size);
        glFlush();
}
int main(int argc, char* argv[])
{
        printf("Enter no. of vertices: \n");
        scanf_s("%d", &poly_size);
        org_poly_size = poly_size;
        for (int i = 0; i < poly_size; i++)
        {
                printf("Polygon Vertex:\n");
                scanf_s("%d%d", &poly_points[i][0], &poly_points[i][1]);
                org_poly_points[i][0] = poly_points[i][0];
                org_poly_points[i][1] = poly_points[i][1];
        }

        printf("Enter no. of vertices of clipping window:");
        scanf_s("%d", &clipper_size);
        for (int i = 0; i < clipper_size; i++)
        {
                printf("Clip Vertex:\n");
                scanf_s("%d%d", &clipper_points[i][0], &clipper_points[i][1]);
        }

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

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

Experiment No..... Page No.....

```
if (i_pos >=0 && k_pos >=0)
{ newpoints[new_poly_size][0]=kx; newpoints[newpolysize][1]=ky;
  new_polysize++; }
else if (i_pos<0 && k_pos>=0)
{ new_points[new_poly_size][0]=x_intersect(x1,y1,x2,y2,ix,iy,
  newpoints[new_poly_size][1]=y_intersect(x1,y1,x2,y2,ix,iy,
  new_poly_size++;
  newpoints[new_polysize][0]=kx; newpoints[newpolysize][1]=ky;
  newpoly_size++; }
 else if(i_pos >=0 && k_pos<0)
{ newpoints[new_polysize][0]=xintersect(x1,y1,x2,y2,ix,iy,kx,ky
  new_points[new_poly_size][1]=yintersect(x1,y1,x2,y2,ix,iy,kx,ky
  newpolysize++;
 }
 else {}
{ polysize=newpolysize;
  for(int i=0; i<polysize;i++)
  { polypoints[i][0]=newpoints[i][0];
    polypoints[i][1]=newpoints[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()
{ int i; glColor3f(1,0,0); drawpoly(clippoints,clippoint);
  for(int i=0;i<clippersize;i++)
  { int k=(i+1)%clippersize;
```



Name of Experiment .....

Experiment No. .....

```
clip(polypoints, polysize, clippoints[i][0], clippoints[i][1],
    clippoints[k][0], clippoints[k][1]); }
glColor3f(0,0,1);
drawPoly(polypoints, polysize);
glFlush(); }
int main(int argc, char *argv)
{ printf("Enter no of vertices");
  scanf("%d", &polysize);
  for(int i=0; i<polysize; i++)
  { printf("Polygon vertex");
    scanf("%d %d", &polypoints[i][0], &polypoints[i][1]);
    org_polypoints[i][0] = polypoints[i][0];
    org_polypoints[i][1] = polypoints[i][0];
  }
for(int i=0; i<clippersize; i++)
{ printf("clip vertex:"); scanf("%d", &clippers[i][0], &clippers[i]
  {
glutInit(&argc, argv)
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB)
glutInitWindowSize(400, 400);
glutWindowPosition(100, 100);
glutCreateWindow(display);
  glutMainLoop();
  return 0;
}
```

```
#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, 0.0);
        glCallList(CAR);
        glPushMatrix();
        glTranslatef(25, 25, 0.0);    //move to first wheel position
        //draw_wheel();
        glCallList(WHEEL);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(75, 25, 0.0);    //move to 2nd wheel position
        ////draw_wheel();
        glCallList(WHEEL);
        glPopMatrix();
        glFlush();
}
void myDisp() {
        glClear(GL_COLOR_BUFFER_BIT);
        carlist();
```

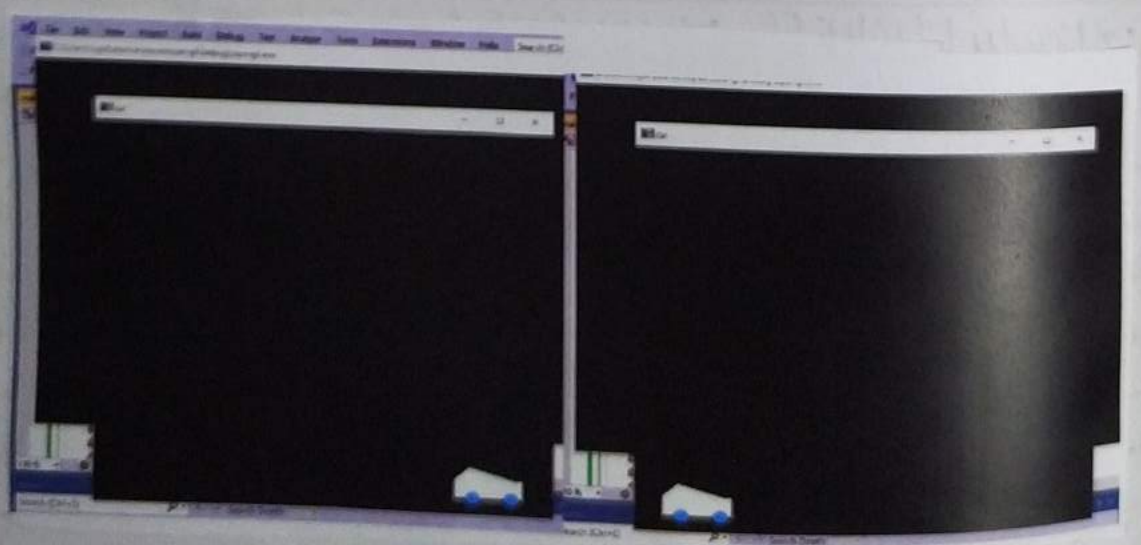
Name of Experiment ..... Model a car using display list ..... Date .....

Experiment No. 09 ..... Page No. .....

Write a program to model a car using display list move it from one end to another control speed with mouse.

```
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#define CAR 1
#define WHEEL 2.
float s=1;
void carlist() {
    glNewList(CAR, GL_COMPILE);
    glColor3f(1,1,1); glBegin(GL_POLYGON);
    glVertex3f(0,25,0); glVertex3f(90,25,0); glVertex3f(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);
    glSolidSphere(10,25,25);
    glEndList(); }
void MyKeyboard(unsigned char key, int x, int y) {
    switch(key) { case 'r': GlutPostRedisplay(); break;
                  case 'q': exit(0); default: break; } }
void myinit()
{ glClearColor(0,0,0,0) glOrtho(0,600,0,600,0,600); }
void movecar(float s)
{ glTranslatef(s,0,0) glCallList(CAR); glPushMatrix();
  glTranslate(25,25,0);
```

```
        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("car");
        myInit();
        glutDisplayFunc(myDisp);
        glutMouseFunc(mouse);
        glutKeyboardFunc(mykeyboard);
        glutMainLoop();
}
```



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

Experiment No..... Page No.....

```
glCallList (WHEEL);
glPopMatrix() ; glPushMatrix(); glTranslatef (75,25,0);
glCall list (WHEEL); glPopMatrix(); glFlush; }
void Mydisp() {
    glClear (GL_COLOR_BUFFER_BIT);
     carlist ();
     movecar (&);
     wheellist(); }
void mouse (int btn, int state, int x, int y) {
 if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
        St = 5; myDisp(); }
else if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
        St = 2; 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();
}
```

```
#include <stdlib.h>
#include <GL/glut.h>
#include<gl\GL.h>
#include<gl\GLU.h>
#include <time.h>
GLfloat vertices[] = { -1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0, -1.0,1.0,-1.0, -1.0,-1.0,1.0,1.0,-1.0,1.0, 1.0,1.0,1.0, -1.0,1.0,1.0 };
GLfloat normals[] = { -1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0, -1.0,1.0,-1.0, -1.0,-1.0,1.0,1.0,-1.0,1.0, 1.0,1.0,1.0, -1.0,1.0,1.0 };
GLfloat colors[] = { 0.0,0.0,0.0,1.0,0.0,0.0,0.0,1.0,1.0,0.0,0.0, 0.0,1.0,0.0, 0.0,0.0,1.0,1.0,0.0,0.0,1.0, 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();
}
void spinCube()
{
        delay(0.01);
        theta[axis] += 2.0;
        if (theta[axis] > 360.0) theta[axis] -= 360.0;
        glutPostRedisplay();
}
void mouse(int btn, int state, int x, int y)
{
        if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
        if (btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
        if (btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
}
void myReshape(int w, int h)
{
        glViewport(0, 0, w, h);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        if (w <= h)
                glOrtho(-2.0, 2.0, -2.0 * (GLfloat)h / (GLfloat)w,
                        2.0 * (GLfloat)h / (GLfloat)w, -10.0, 10.0);
        else
                glOrtho(-2.0 * (GLfloat)w / (GLfloat)h,
                        2.0 * (GLfloat)w / (GLfloat)h, -2.0, 2.0, -10.0, 10.0);
        glMatrixMode(GL_MODELVIEW);
}
Void main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowPosition(100, 100);
        glutInitWindowSize(500, 500);
        glutCreateWindow("colorcube");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(displaySingle);
```

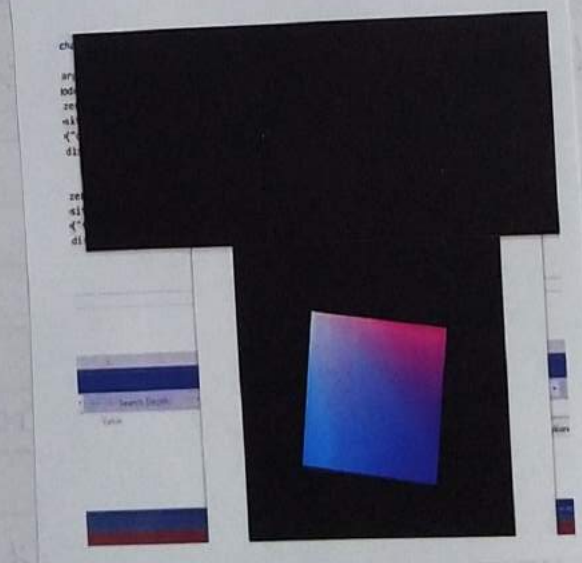
Name of Experiment: Colour Cube Date:

Experiment No.: 10 Page No.:

Create a color cube and spin it using OPENGL Transformation

```
#include<stdlib.h>
#include<GL/glut.h>
#include<gl/gl.h>
#include<GL/glu.h>
#include<time.h>
GLfloat vertices[] = {-1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0,
-1.0,1.0,-1.0,-1.0,-1.0,1.0,1.0,-1.0,1.0,1.0,1.0,1.0,-1.0,1.0,1.0};
GLfloat normal[] = {-1.0,-1.0,-1.0,1.0,-1.0,-1.0,1.0,1.0,-1.0,-1.0,
1.0,-1.0,-1.0,-1.0,1.0,1.0,-1.0,-1.0,1.0,1.0,1.0,1.0,1.0,1.0};
GLfloat colors[] = {0.0,0.0,0.0,1.0,0.0,0.0,1.0,1.0,0.0,0.0,0.0,1.0,
0.0,0.0,0.0,1.0,1.0,0.0,1.0,1.0,1.0,1.0,0.0,1.0,1.0,1.0};
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 delta[] = {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(1.1,1.1) glEnd();
glFlush();
}
```

```
    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();
}
```



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

Experiment No..... Page No.....

```
void myReshape (int w, int h)
{ glViewPort (0,0,w,h) glMatrixMode (GL_PROJECTION)
  glLoadIdentity(); if(w<=h) glOrtho(-2.0,2.0,-2.0*(GLfloat)h/
  (GLfloat)w, 2.0*(GLfloat)h/(GLfloat)w, -10.0, 10.0);
  else
    glOrtho(-2.0*(GLfloat)w/(GLfloat)h, 2.0*(GLfloat)w/
  (GLfloat)h, -2.0, 2.0, -10.0, 10.0);
  glMatrixMode(GL_MODEL_VIEW) }
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==GLUTDOWN) axis=1;
  if(btn==GLUT_RIGHT_BUTTON && State==GLUTDOWN) axis=2; }
void main(int a, char **v)
{ glutInit(&a,v); glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(100,100); glutWindowSize(500,500);
  glutCreateWindow("colorcube");
  glutReshapeFunc(myreshape); glutdisplayfunc(displaySingle);
  glutIdleFun(spinCube);
  glutMouseFunc(mymouse);
  glEnable(GL_DEPTH_TEST)
  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(1,1,1); glutMainLoop()
}
```

```
#include<gl/glut.h>
#include<math.h>
#include<stdio.h>
struct screenPt {
        int x;
        int y;
};
typedef enum { limacon = 1, cardioid = 2, threeLeaf = 3, spiral = 4 } curveName;
int w = 600, h = 500;
int curve = 1;
int red = 0, green = 0, blue = 0;
void myinit(void) {
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(0.0, 200.0, 0.0, 150.0);
}
void lineSegment(screenPt p1, screenPt p2) {

        glBegin(GL_LINES);
        glVertex2i(p1.x, p1.y);
        glVertex2i(p2.x, p2.y);
        glEnd();
        glFlush();
}
void drawCurve(int curveNum) {
        const double twoPi = 6.283185;
        const int a = 175, b = 60;
        float r, theta, dtheta = 1.0 / float(a);
        int x0 = 200, y0 = 250;
        screenPt curvePt[2];
        curve = curveNum;
        glColor3f(red, green, blue);
        curvePt[0].x = x0;
        curvePt[0].y = y0;
        glClear(GL_COLOR_BUFFER_BIT);
        switch (curveNum) {
        case limacon: curvePt[0].x += a + b; break;
        case cardioid: curvePt[0].x += a + a; break;
        case threeLeaf: curvePt[0].x += a; break;
        case spiral: break;
        default: break;
        }
        theta = dtheta;
        while (theta < twoPi) {
                switch (curveNum) {
                case limacon: r = a * cos(theta) + b; break;
                case cardioid: r = a * (1 + cos(theta)); break;
                case threeLeaf: r = a * cos(3 * theta); break;
                case spiral: r = (a / 4.0) * theta; break;
                default: break;
                }
                curvePt[1].x = x0 + r * cos(theta);
                curvePt[1].y = y0 + r * sin(theta);
                lineSegment(curvePt[0], curvePt[1]);

                curvePt[0].x = curvePt[1].x;
                curvePt[0].y = curvePt[1].y;
                theta += dtheta;
        }
}

void colorMenu(int id) {
        switch (id) {

        case 0:
                break;
```

Name of Experiment MENU with curves. Date

Experiment No. 11 Page No.

Create a menu with curves named as curves, colors & quit.

```
#include<gl/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; int red=0,green=0, blue=0;
void myinit(void){
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 = 1.0/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 cardiod: curvePt[0].x+ =a+a; break;
case threeleaf: curvePt[0].x + =a; break;
case spiral: break;
default : break;
}
```

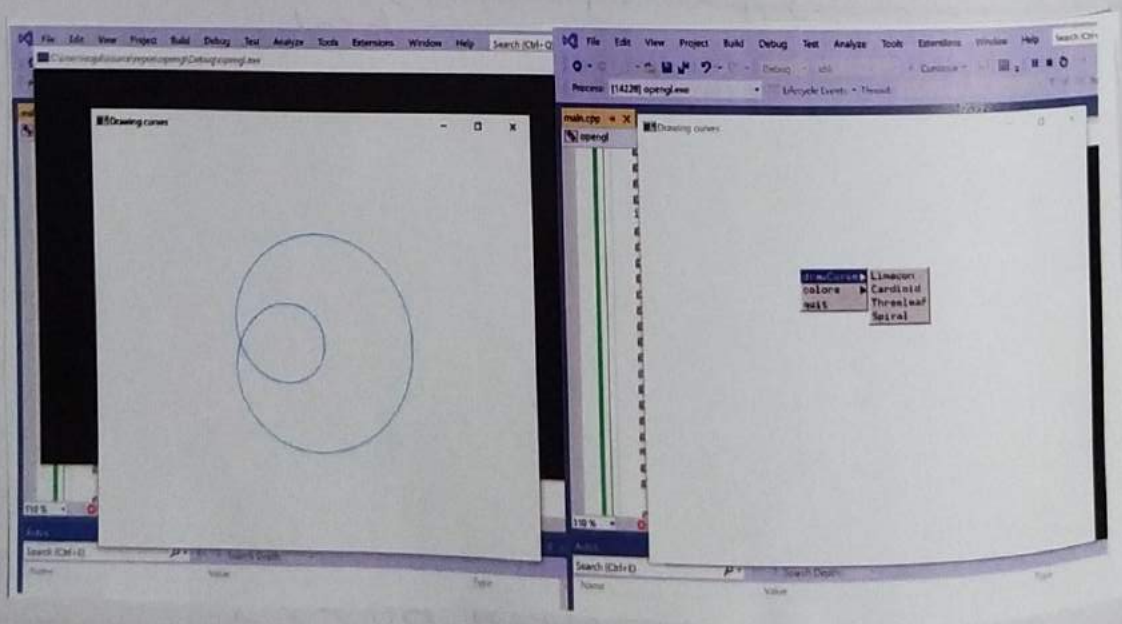
```
		case 1:
			red = 0;
			green = 0;
			blue = 1;

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

		case 4:
			red = 1;
			green = 0;
			blue = 0;

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

			break;
		case 5:
			red = 1;
			green = 0;
			blue = 1;
			break;
		case 6:
			red = 1;
			green = 1;
			blue = 0;
			break;
		case 7:
			red = 1;
			green = 1;
			blue = 1;
			break;
		default:
			break;

		}
		drawCurve(curve);
}
void main_menu(int id) {

		switch (id) {

		case 3: exit(0);
		default: break;
		}
}
void mydisplay() {
		/*int curveNum=1;
		glClear(GL_COLOR_BUFFER_BIT);
		/*printf("Enter the integer value corresponding to one of the followinf curve names:\n");
		printf("1 - limacon\n2 - cardioid\n3 - threeleaf\n4 - spiral\n");
		scanf_s("%d", &curveNum);*/.

		/*if (curveNum == 1 || curveNum == 2 || curveNum == 3 || curveNum == 4)
		drawCurve(curveNum);*/

}

void myreshape(int nw, int nh) {
```

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

Experiment No..... Page No.....

```
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; brea; default : break; }
curvept[i].x = x0 + r*cos(theta);
curvept[i].y = y0 + r*sin(theta);
linesegment(curvept[0], curvept[i]);
curvept[0].x = curvept[i].x; curvept[0].y = curvept[i].y;
theta += dtheata; } }
void colorMenu(int id) {
switch(id) { case 0: break; case 1: red=0; gree=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; gren=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; grin=1; blue=1; break;
default: break; } drawcurve(curve); }
void main_menu(int id) {
  switch(id) { case 3: exit(0); default: break; } }
void mydisplay() { }
void myreshape(int nw, int nh)
{ glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0, (double)nw, 0.0, (double)nh);
  glClear(GL_COLOR_BUFFER_BIT); }
```

```
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, (double)nw, 0.0, (double)nh);
        glClear(GL_COLOR_BUFFER_BIT);
}

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



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

Experiment No..... Page No.....

```
void main ( int a, char** v)
{ glutInit (&a, v)
 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 ("Threeleg", 3);
 glutAddMenuEntry ("Spiral", 4);
 glutAttachMenu (GLUT_LEFT_BUTTON)
 int colorid = glutCreateMenu (colorsmenu);
 glutAddMenuEntry ("Red", 4);
 glutAddMenuEntry ("Green", 2);
 glutAddMenuEntry ("Blue", 1);
 glutAddMenuEntry ("Black", 0)
 glutAddMenuEntry ("Yellow", 6)
 glutAttachMenu (GLUT_LEFT_BUTTON);
 glutCreateMenu (main_menu);
 glutAddSubMenu ("Draw curve", curveid);
 glutAddSubMenu ("colors", colorid);
 glutAddMenuEntry ("quit", 3);
 glutAttachMenu (GLUT_LEFT_BUTTON);
 myinit();
 glutDisplayFunc (mydisplay);
 glutReshapeFunc (myreshape);
 glutMainLoop()
}
```

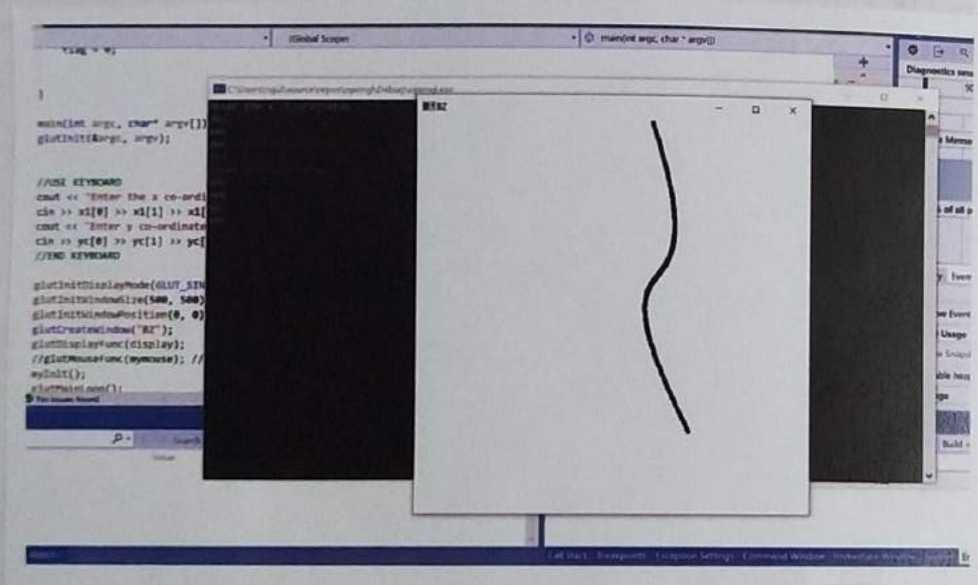
```
#include<iostream>
#include<math.h>
#include<gl/glut.h>
using namespace std;
float f, g, r, x1[4], yc[4];
int flag = 0;
void myinit() {glClearColor(1, 1, 1, 1);
        glColor3f(1, 1, 1);
        glPointSize(5);
        gluOrtho2D(0, 500, 0, 500);}
void drawPixel(float x, float y) {
        glBegin(GL_POINTS);
        glVertex2f(x, y);
        glEnd();
}
void display() {
        glClear(GL_COLOR_BUFFER_BIT); int i;
        double t;
        glColor3f(0, 0, 0);
        glBegin(GL_POINTS);
        for (t = 0; t <= 1; t = t + 0.005) {
                double xt = pow(1 - t, 3) * x1[0] + 3 * t * pow(1 - t, 2) * x1[1] + 3 * pow(t, 2) * (1 - t) * x1[2] + pow(t, 3) * x1[3];
                double yt = pow(1 - t, 3) * yc[0] + 3 * t * pow(1 - t, 2) * yc[1] + 3 * pow(t, 2) * (1 - t) * yc[2] + pow(t, 3) * yc[3];
                glVertex2f(xt, yt);
        }
        glColor3f(1, 1, 0);
        for (i = 0; i < 4; i++) {
                glVertex2f(x1[i], yc[i]);
                glEnd();
                glFlush();
        }
}
void mymouse(int btn, int state, int x, int y)
{
        if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN && flag < 4)
        {
                x1[flag] = x;
                yc[flag] = 500 - y;
                cout << " X: " << x << " Y" << 500 - y;
                glPointSize(3);
                glColor3f(1, 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);
        cout << "Enter the x co-ordinates";
        cin >> x1[0] >> x1[1] >> x1[2] >> x1[3];
        cout << "Enter y co-ordinates";
        cin >> yc[0] >> yc[1] >> yc[2] >> yc[3];
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("BZ");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

Name of Experiment ..... Bezier curve ..... Date .....

Experiment No. 12 ..... Page No. .....

write a program to construct Bezier curve.

```
#include <iostream.h>
#include <math.h>
#include<gl/glut.h>
using namespace std;
float f, g, r, x1[4], yc[4]; int flag=0;
void myinit() {
 glClearColor(1,1,1,1);
 glColor3f(1,1,1);
 glPointSize(5);
 glOrtho2D(0,500,0,500); }
void drawPixel(float x,float y)
{ glBegin(GL_POINTS) glVertex2f(x,y); glEnd();}
void display() {
glClear(GL_COLOR_BUFFER_BIT); int i; double t;
glColor3f(0,0,0); glBegin(GL_POINTS);
for(t=0; t<1; t=t+0.005) {
 double xt=pow(1-t,3)* x1[0]+ 3*t*pow(1-t,2)*x1[1]
 + 3*pow(t,2)*(1-t)*x1[2] + pow(t,3)*x1[3];
 double yt= pow(1-t,3)*yc[0] +3*t*pow(1-t,2)*yc[1]
 +3*pow(t,2)*(1-t)*yc[2]+pow(t,3)*yc[3];
 glVertex2f(xt,yt);
}
glColor3f(1,1,0);
 for(i=0; i<4,i++) {
    glVertex2f(x1[i],yc[i]);
    glEnd();
    glFlush(); }
```


(Global Scope)
main(int argc, char * argv[])
BZ

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

Experiment No. ..... Page No. .....

```
void mymouse(int btn, int state, int x, int y)
{ if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN && flag<4)
  {
    xc[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 && button == GLUT_LEFT_BUTTON)
  { glColor3f(0,0,1); display(); flag=0 }
}

int main(int a, char** v)
{ glutInit(&a, v)
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500)
  glutInitWindowPosition(0, 0)
  glutCreateWindow("B2");
  glutDisplayFunc(display);
  myinit();
  glutMainLoop()
}
```