Name of the Institute R.V. COLLEGE of ENGINEERING

Laboratory Certificate

This is to certify that Smt./Sri. SATHVIK .K.R

has satisfactorily completed the course of Experiments in Practical COMPUTER GRAPHICS Prescribed by the CSE DEPT. VTU University in the Laboratory of this College in the year 20 -20

Signature of the Teacher in charge of the batch

Date:

Head of the Department

Name of the Candidate SATHVIK . K.R

Reg No. 1RV17CS141

Examination Centre

Date of Practical Examination.

The Candidate is expected to retain his/her journal till he/she passes in the subject

INDEX

Name. SATHVIK. K. R

Class. 7th Sem C Year IV

Expt. No.	Date	Title of the Experiments	Page No.	Date of Submission	Remarks
1.	24/10	Generating Line using Bresenham's Line drawing	1-3	24/10/20	
2	24/10	Generating Circle and ellipse using Bresenham's mid point technique	4-6	24/10/20	
3	24/10	Generating 3D Sepenski gasket.	7-8	24/10/20	
4	24/10	Filling polygon using scan line area fill algorithm	9-10	24/10/20	
5	8/11	Creating house and rotating about given point and y=mx+c	11-13	8/11/20	
6	25/11	Implementing Cohen-Sutherland line clipping algorithm	14-16	25/11/20	

INDEX

Name. Sathvik. K. R

Class. 7th Sem C Year IV

Expt. No.	Date	Title of the Experiments	Page No.	Date of Submission	Remarks
7	25/11	Implementing Liang-Barsky line clipping algorithm	17-19	25/11/20	
8	25/11	Implementing the Cohen-Hogeman polygon Clipping algorithm	20-22	25/11/20	
9	8/12	Model a Car using display list and move	23-24	8/12/20	
10	8/12	Create a Color cube and spin it using GL transformation	25-26	8/12/20	
11	26/12	Program to create menu and appropriate service	27-30	26/12/20	
12	26/12/20	Program to construct Bezier curve	31-33	26/12/20	

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

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

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

}

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

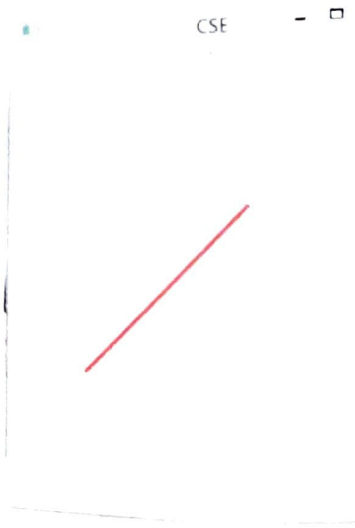
PAGE NO : 1
DATE : 24/10
EXP.NO. : 01

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

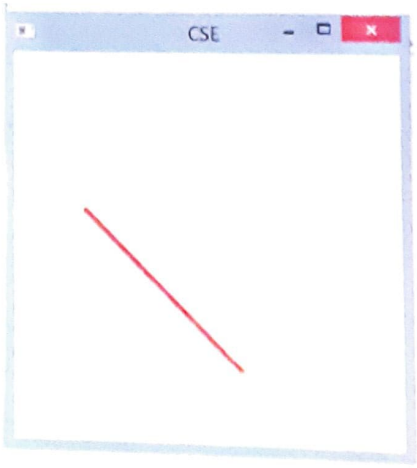
```
#include<iostream.h>
#include <GL/glut.h>
#include <time.h>

using namespace std;
int x1, x2, y1, y2;
void draw_pixel (int x, int y)
{   glColor3f (1,0,0)
        glBegin ( GL_POINTS);
        glVertex2i (x,y);
        glEnd();
        glFlush();
}

void Drawline ()
{   int dx, dy; i, e, incx, incy, inc1, inc2, x, y;
    dx = x2-x1 ;   dy = y2-y1 ;
    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 = y1 ;
    if (dx > dy) {  draw_pixel (x,y);
```

ARUN'S

```
void
{
```




```
}
voic
{}
int
{
```




```
}
```

```
void display () { }
int main (int ac, char* argv)
{
    glutInit (&ac, av);
    glutInitDisplay ( GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize (500, 500);
    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;

//Circle
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();
}

//Generate circle using Bresenham's circle drawing algorithm
void circlebres()
{
	glClear(GL_COLOR_BUFFER_BIT);
	int x = 0, y = r;
	int d = 3 - 2 * r;
	while (x <= y)
	{
		draw_circle(xc, yc, x, y);
		x++;
		if (d < 0)
			d = d + 4 * x + 6;
		else
		{
			y--;
			d = d + 4 * (x - y) + 10;
		}
		draw_circle(xc, yc, x, y);
	}
	glFlush();
}
int p1_x, p2_x, p1_y, p2_y;
int point1_done = 0;
void myMouseFunccircle(int button, int state, int x, int y)
{
	if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN && point1_done == 0)
	{
		p1_x = x - 250;
		p1_y = 250 - y;
		point1_done = 1;
	}
	else if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
	{
		p2_x = x - 250;
		p2_y = 250 - y;
		xc = p1_x;
		yc = p1_y;
		float exp = (p2_x - p1_x) * (p2_x - p1_x) + (p2_y - p1_y) * (p2_y - p1_y);
		r = (int)(sqrt(exp));
		circlebres();
		point1_done = 0;
	}
}
//ELLIPSE
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();
}

//Generate ellipse using Bresenham's ellipse drawing algorithm
void ellipsebres()
{
	glClear(GL_COLOR_BUFFER_BIT);
	float dx, dy, d1, d2, x, y;
	x = 0;
	y = ry;
```

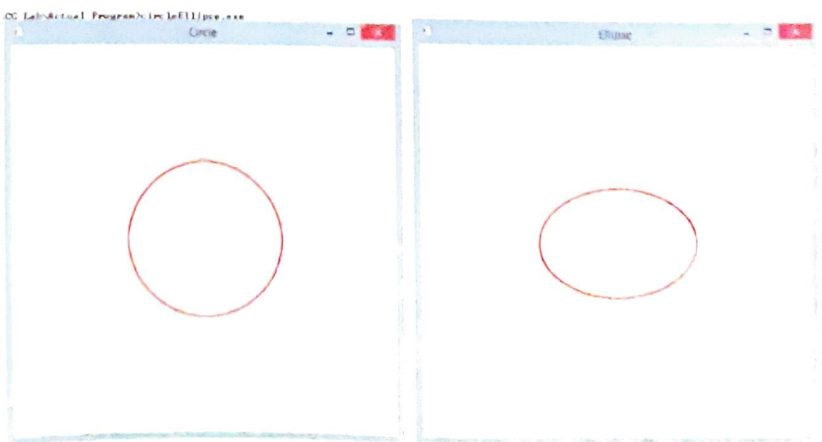
PAGE NO : 24/10
DATE : 4
EXP.NO. : 02

write a program to generate Bircle and ellinse using Bresenham's Circle and Ellinse

```
#include <gl/glut.h>
#include <stdio.h>
#include <math.h>
int xc, yc, r, rx, ry, xce, yce;
int drawCircle (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);
    glVertex 2i (xc+y, yc+y); glVertex2i( xc-y, yc+y);
    glVertex 2i (xc+y, yc-y); glVertex2i (xc-y, yc-y);
    gl.End (); }
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; point_done = 0;
void draw_ellinse (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);
```

ARUN'S

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

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



```
switch (ch) {
case 1 : Printf (" Enter co-oridnates of center of circle );
    Scanf (" %d %d %d ", &xc, &yc, &r);
    glut CreateWindow( "Circle");
    glut DisplayFunc (bres_Circle); break ;
case 2 : Printf (" Enter Coordinates of ellipse );
    Scanf (" %d %d %d %d", &xc, &yc, &rx, &ry);
    glut Create Window(" ellipse");
    glut Display Func (mid_ellipse); break ;
}
minit (); glut MainLoop();
}
```

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

void draw_tetra(point a, point b, point c, point d) {
        glColor3f(0.0, 0.0, 0.0);
        draw_triangle(a, b, c);
        glColor3f(1.0, 0.0, 0.0);
        draw_triangle(a, c, d);
        glColor3f(0.0, 1.0, 0.0);
        draw_triangle(a, b, d);
        glColor3f(0.0, 0.0, 1.0);
        draw_triangle(b, c, d);
}

void draw_triangle(point a, point b, point c) {
        glBegin(GL_TRIANGLES);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
        glEnd();
}

void divide_tetra(point a, point b, point c, point d, int i) {
        point v1, v2, v3, v4, v5, v6;
        if (i > 0) {
                for (int j = 0; j < 3; j++) {
                        v1[j] = (a[j] + b[j]) / 2;
                        v2[j] = (a[j] + c[j]) / 2;
                        v3[j] = (a[j] + d[j]) / 2;
                        v4[j] = (b[j] + c[j]) / 2;
                        v5[j] = (b[j] + d[j]) / 2;
                        v6[j] = (c[j] + d[j]) / 2;

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

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

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

        glFlush();
}

void myinit() {
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1, 1, 1, 1);
        glOrtho(-12.0, 12.0, -12.0, 12.0, -12.0, 12.0);
}
int main(int argv, char** argc)
{
        std::cout << "Enter the number of iterations: ";
        std::cin >> M;
        glutInit(&argv, argc);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(900, 900);
        glutCreateWindow("Sierpinski Gasket");
        glutDisplayFunc(tetrahedron);
        glEnable(GL_DEPTH_TEST);
        myinit();
        glutMainLoop();
}
```

PAGE NO : 7
DATE : 24/10/20
EXP.NO. : 03

Write a program that recursively subdivides a tetrahedron to form 3D - Serpenski 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, v2, m-1);
    divide_triangle (c, v2, v3, m-1);
    divide_triangle (b, v3 v1 m-1);
  } else    draw triangle (a.b.c);
}
```

ARUN'S

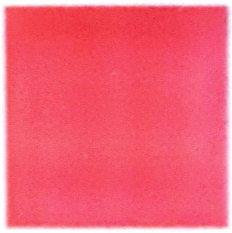
```
#include<stdlib.h>
#include<GL/glut.h>
#include<algorithm>
#include<iostream>
#include<windows.h>
using namespace std;
float x[100], y[100];
int n, m;
int wx = 500, wy = 500;
static float intx[10] = { 0 };
void draw_line(float x1, float y1, float x2, float y2) {
        Sleep(100);
        glColor3f(1, 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 = x2; 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: ");
        scanf_s("%d", &n);
        printf("Enter coordinates of endpoints: ");
        for (int i = 0; i < n; i++)
        {
                printf("X-coord Y-coord: ");
                scanf_s("%f %f", &x[i], &y[i]);
        }
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(0, 0);
        glutCreateWindow("scanline");
        glutDisplayFunc(display_filled_polygon);
        myInit();
        glutMainLoop();

}
```

PAGE NO : 24/10/24
DATE : 8/9
EXP.NO. : 04

write a program to fill a polygon using scan-area filling algorithm

```
#include <gl/glut.h>
#include <iostream.h>
#include <stdlib.h>
#include <windows.h>
#include <algorithm>
using namespace std;
float x[100], y[100]; int n, m, wx = 500, wy = 500,
static float X[10] = {0};

void drawline (float x1, float y1, float x2, float y2)
{  glColor3f (1.0, 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 = x1; x2 = temp;
                 temp = y1; y1 = y2; y2 = temp; }
  if (Scanline > y1 && Scanline < y2)
    int x[m++] = x1 + (scanline - y1) * (x2 - x1) / (y2 - y1);
}
Void ScanFill (float x[], float y[])
{ for (int Si = 0; Si < wy; Si++) { m = 0;
   for (int i = 0; i < n; i++)
     edgeDetect (x[i], y[i], x[(i+1)%n], y[(i+1)%n], Si); }
```

ARUN'S



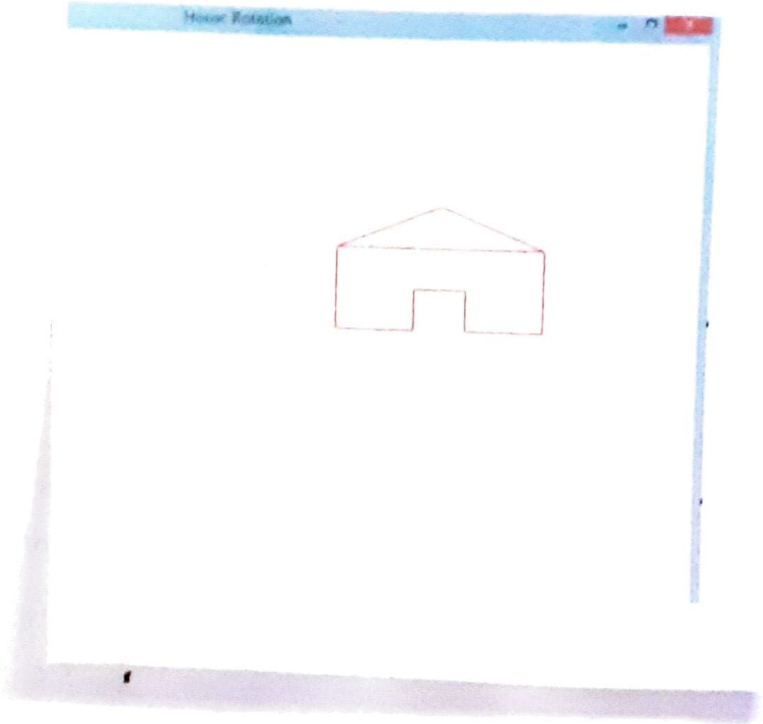
```
#include<gl/glut.h>
#include <math.h>
//#include<stdlib.h>
#include<stdio.h>
//RIGHT CLICK TO SHOW REFLECTED HOUSE
float house[11][2] = { { 100,200 },{ 200,250 },{ 300,200 },{ 100,200 },{ 100,100 },{ 175,100 },{ 175,150 },{ 225,150
},{ 225,100 },{ 300,100 },{ 300,200 } };
int angle;
float m, c, theta;

void display()
{
        glClearColor(1, 1, 1, 0);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-450, 450, -450, 450);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        //NORMAL HOUSE
        glColor3f(1, 0, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glFlush();
        //ROTATED HOUSE
        glPushMatrix();
        glTranslatef(100, 100, 0);
        glRotatef(angle, 0, 0, 1);
        glTranslatef(-100, -100, 0);
        glColor3f(1, 1, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glPopMatrix();
        glFlush();
}

void display2()
{
        glClearColor(1, 1, 1, 0);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(-450, 450, -450, 450);
        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
        //normal house
        glColor3f(1, 0, 0);
        glBegin(GL_LINE_LOOP);
        for (int i = 0; i < 11; i++)
                glVertex2fv(house[i]);
        glEnd();
        glFlush();
        // line
        float x1 = 0, x2 = 500;
        float y1 = m * x1 + c;
        float y2 = m * x2 + c;
        glColor3f(1, 1, 0);
        glBegin(GL_LINES);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
        glEnd();
        glFlush();

        //Reflected
        glPushMatrix();
        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();
```

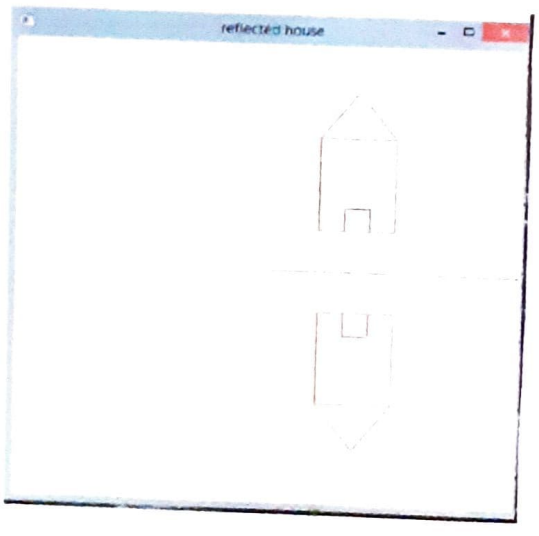
PAGE NO : 11
DATE : 8/11/20
EXP.NO. : 05

Write a Program to Create house like figure and
i) Reflect it about given fixed point using OpenGL
ii) Reflect 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);
    glLoadIdentify();
    glOrtho2D(-450,450,-450,450);
    glMatrixMode(GL_MODEL_VIEW);
    glLoadIdentity();   glColor3f(1.0,0.0);
    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(1,0,0);
    glBegin(GL_LINE_LOOP); for(i=0;i<11;i++)
    glVertex2fv(house[i]); glEnd(); glPopMatrix();
```

ARUN'S


House Rotation


reflected house

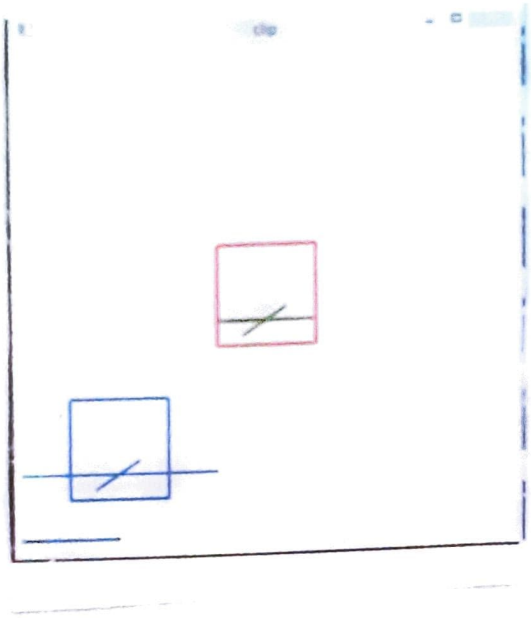
```
#include<stdio.h>
#include<stdlib.h>
#include<gl/glut.h>
#define outcode int
#define true 1
#define false 0
double xmin, ymin, xmax, ymax;
double xvmin, yvmin, xvmax, yvmax;
const int RIGHT = 4;
const int LEFT = 8;
const int TOP = 1;
const int BOTTOM = 2;
int n;
struct line_segment {
        int 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;
                        }
                        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;
```

PAGE NO : 14
DATE : 25/11/20
EXP.NO. : 06

write a program to implement cohen-Sutherland line clipping algorithm

```
#include <stdio.h>
#include <stdlib.h>
#include <gl/glut.h>
# define outcode int
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 line_segment 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 00, 01, 0out; bool accept = false;
  bool done = false;
    00 = ComputeCode (x0, y0);
    01 = computeCode (x1, y1);
do { if (! (00 | 01)) { accept = true; done = true; }
  else if (00 & 01) done = true;
  else {
```

ARUN'S



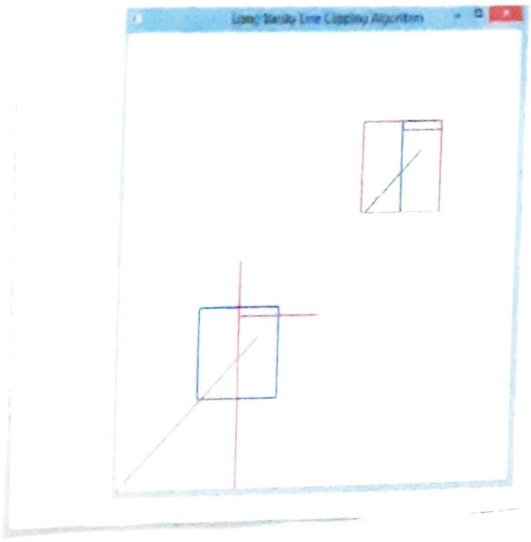
```
#include <stdio.h>
#include <GL/glut.h>

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

int n;

struct line_segment {
        int x1;
        int y1;
        int x2;
        int y2;
};
struct line_segment ls[10];
int cliptest(double p, double q, double* u1, double* 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);
                                        double vx0 = xvmin + (x0 - xmin) * sx;
                                        double vy0 = yvmin + (y0 - ymin) * sy;
                                        double vx1 = xvmin + (x1 - xmin) * sx;
                                        double vy1 = yvmin + (y1 - ymin) * sy;

                                        glColor3f(0.0, 0.0, 1.0); // draw blue colored clipped line
                                        glBegin(GL_LINES);
                                        glVertex2d(vx0, vy0);
                                        glVertex2d(vx1, vy1);
                                        glEnd();
                                }
}// end of line clipping

void display()
{
```

PAGE NO : 17
DATE : 25/11
EXP.NO. : 07

Write a program to implement Liang Barsky Algorithm

```
#include <stdio.h>
#include <GL/glut.h>
#include <iostream>
using namespace std; int n;
double xvmin, yvmin, xvmax, yvmax, xmin, ymin, xmax, ymax;
struct Line_Segment { int x1, x2, y1, y2; } ls[10];
int clipTest (double p, double q, double *u1, double *u2) {
  double r = q / p;
  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 LiangB (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))
```

ARUN'S


Liang Barsky Line Clipping Algorithm

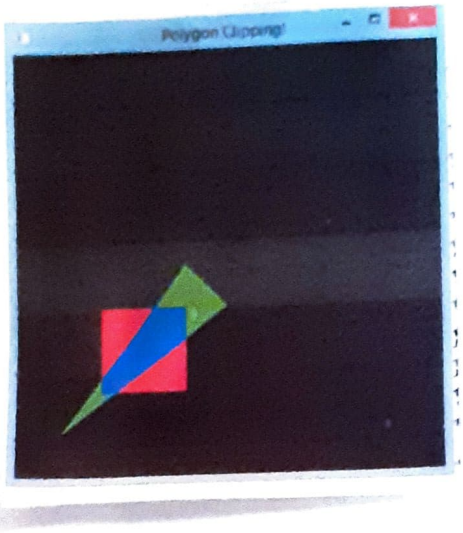
```
#include<iostream>
#include<GL/glut.h>
using namespace std;
int poly_size, poly_points[20][2], org_poly_size, org_poly_points[20][2], clipper_size, clipper_points[20][2];
const int MAX_POINTS = 20;

// 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) * (x3 - x4) -
                (x1 - x2) * (x3 * y4 - y3 * x4);
        int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
        return num / den;
}

// Returns y-value of point of intersectipn of two lines
int y_intersect(int x1, int y1, int x2, int y2,
        int x3, int y3, int x4, int y4)
{
        int num = (x1 * y2 - y1 * x2) * (y3 - y4) -
                (y1 - y2) * (x3 * y4 - y3 * x4);
        int den = (x1 - x2) * (y3 - y4) - (y1 - y2) * (x3 - x4);
        return num / den;
}

// This functions clips all the edges w.r.t one clip
// edge of clipping area
void clip(int poly_points[][2], int& poly_size,
        int x1, int y1, int x2, int y2)
{
        int new_points[MAX_POINTS][2], new_poly_size = 0;

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

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

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

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

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

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

PAGE NO : 20
DATE : 25/11
EXP.NO. : 08

Write a program to implement Cohen - Hodgeman Algorithm

```
#include <iostream>
#include <gl/glut.h>
using namespace std;
int polySize, poly-points[20][2], org-poly-size,
    org-poly-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][i]); 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, y2, x3, y3, x4, y4)
{ int num = (x1*y1 - 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 polyP[][2], int ps, int x1, int y1, int x2, int y2)
{ int new-P[max-points][2]; newPS = 0;
  for (int i=0; i<ps; i++)
  { int k = (i+1)%ps; int ix = polyP[i][0];
    int iy = polyP[i][i]; int kx = polyP[k][0], ky = polyP[k][i];
```

ARUN'S


Polygon Clipping!

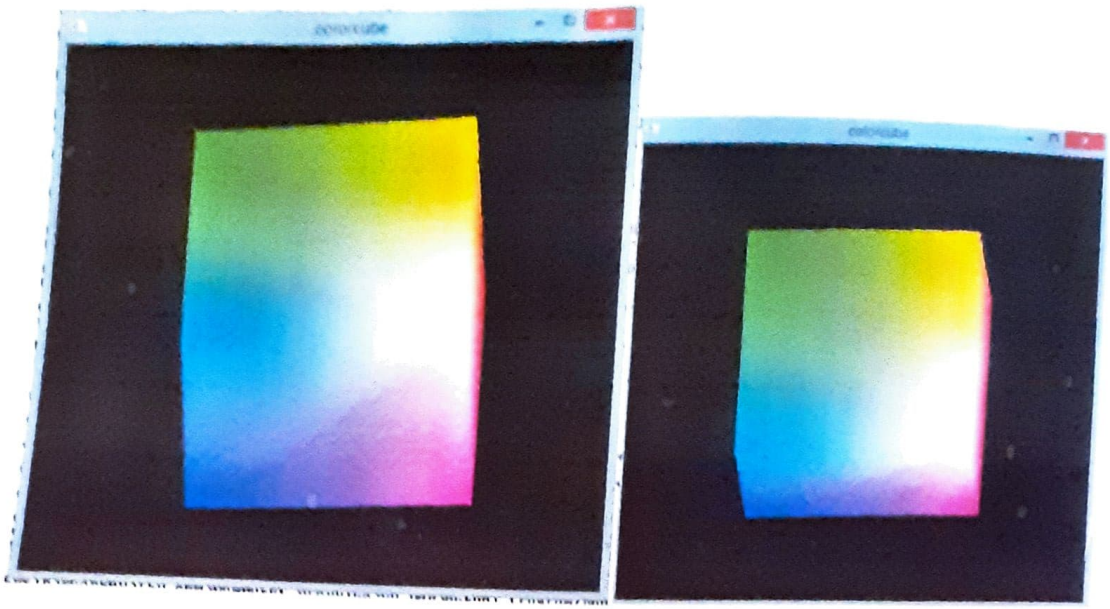
```
#include<GL/glut.h>
#include<math.h>
#include<stdio.h>
#define CAR 1
#define WHEEL 2
float s = 1;
void carlist() {
        glNewList(CAR, GL_COMPILE);
        glColor3f(1, 1, 1);
        glBegin(GL_POLYGON);
        glVertex3f(0, 25, 0);
        glVertex3f(90, 25, 0);
        glVertex3f(90, 55, 0);
        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
        glCallList(WHEEL);
        glPopMatrix();
        glPushMatrix();
        glTranslatef(75, 25, 0.0);     //move to 2nd wheel position
        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("car");
        myInit();
        glutDisplayFunc(myDisp);
        glutMouseFunc(mouse);
        glutKeyboardFunc(mykeyboard);
        glutMainLoop();
}
```

PAGE NO 23
DATE 8/12
EXP NO 09

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; } }
```

ARUN'S



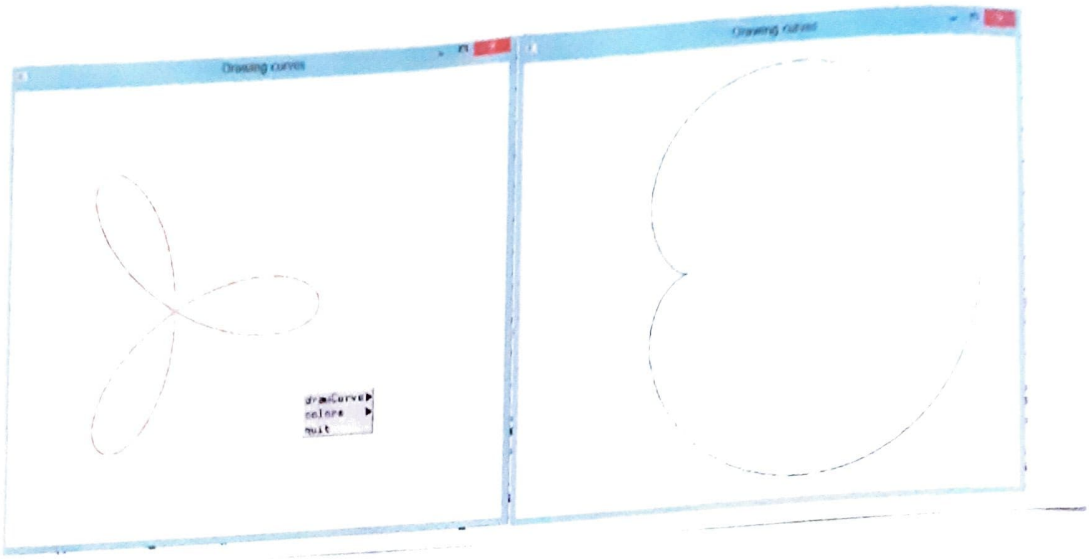
```
#include<gl/glut.h>
#include<math.h>
#include<stdio.h>
struct screenPt {
        int x;
        int y;
};
typedef enum { limacon = 1, cardioid = 2, threeLeaf = 3, spiral = 4 } curveName;
int w = 600, h = 500;
int curve = 1;
int red = 0, green = 0, blue = 0;
void myinit(void) {
        glClearColor(1.0,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

PAGE NO : 27
DATE : 26/12/20
EXP.NO. : 11

Write a program to create a menu with entries named as Curves, colors & quit.

```
#include <gl/glut.h>
#include <math.h>
#include <stdio.h>
Struct SPt {int x; int y;};
typedef enum { limcon=1, Cardiod=2, Threeleaf=3, Spiral} 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 (SPt P1, SPt P2) { glBegin(GL_LINES);
glVertex2i (P1.x, P1.y); glVertex2i (P2.x, P2.y); glEnd();
glFlush(); }
Void DrawCurve (int Curvenum) { const double twoPi=6.28;
const int a=175; b=60; float x; theta, dtheta=1.0f;
int x0=200; y0=250; SPt CurvePt[2]; curve=curveNum;
glColor3f (red, green, blue); CurvePt[0].x = x0;
curvePt[0].y = y0;
    glClear (GL_COLOR_BUFFER_BIT);
    Switch (CurveNum) {
        Case limanon: curvePt[0].x += a+b; break;
        Case cardiod: curvePt[0].x += a+a; break;
```

ARUN'S


Drawing curves
drawCurve
colors
quit
Drawing curves

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

PAGE NO : 31
DATE : 26/12/20
EXP.NO. : 12

Write a program to construct Bezier Curve

```
#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);
```