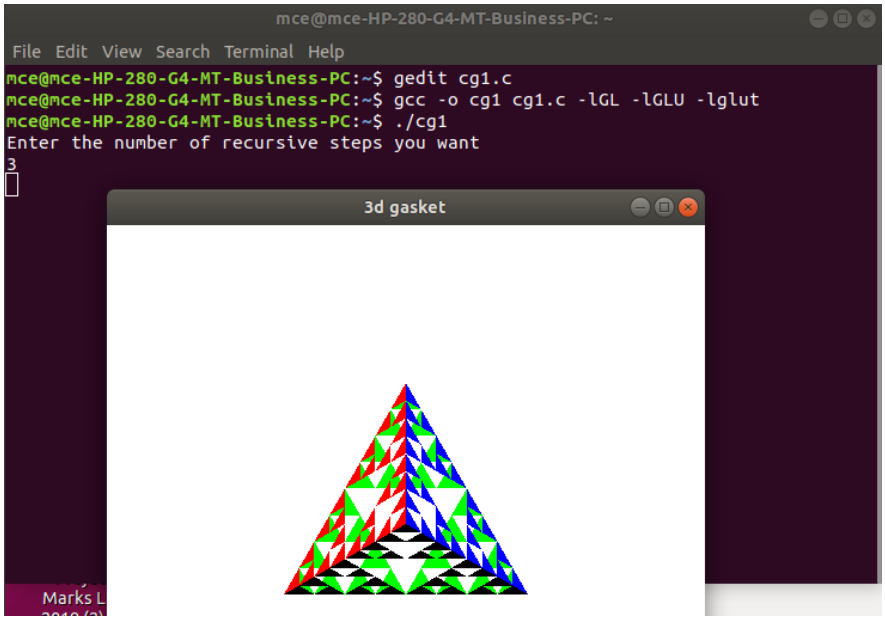
1. Write a C program to recursively subdivide a tetrahedron to form 3D Sierpinski gasket. The number of recursive steps is to be specified by the user.

```
#include <GL/glut.h>
#include <stdlib.h>
#include<stdio.h>
typedef   float  point[3];

point v[]={ {0.0,0.0,1.0},
            {0.0,0.943,-0.33},
            {-0.816,-0.471,-0.33},
            {0.816,-0.471,0.33}};

int n;
void triangle(point a,point b,point c)
{
        glBegin(GL_POLYGON);
        glNormal3fv(a);
        glVertex3fv(a);
        glVertex3fv(b);
        glVertex3fv(c);
        glEnd();
}

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

                        for(j=0;j<3;j++)
                        v2[j]=(a[j]+c[j])/2;

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

                divide_tri(a,v1,v2,m-1);
                divide_tri(c,v2,v3,m-1);
                divide_tri(b,v3,v1,m-1);
        }
```

```
        else
                triangle(a,b,c);
}

void tetrahedron(int m)
{
        glColor3f(1.0,0.0,0.0);
        divide_tri(v[0],v[1],v[2],m);
        glColor3f(0.0,1.0,0.0);
        divide_tri(v[3],v[2],v[1],m);
        glColor3f(0.0,0.0,1.0);
        divide_tri(v[0],v[3],v[1],m);
        glColor3f(0.0,0.0,0.0);
        divide_tri(v[0],v[2],v[3],m);
}


void display(void)
{
        glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        tetrahedron(n);
        glFlush();
}


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);
        glutPostRedisplay();
}


int  main(int argc,char **argv)
{
        printf("Enter the number of recursive steps you want\n");
```

```
        scanf("%d", &n);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB|GLUT_DEPTH);
        glutInitWindowSize(500,500);
        glutCreateWindow("3d gasket");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glEnable(GL_DEPTH_TEST);
        glClearColor(1.0,1.0,1.0,1.0);
        glutMainLoop();
}
```

Output:



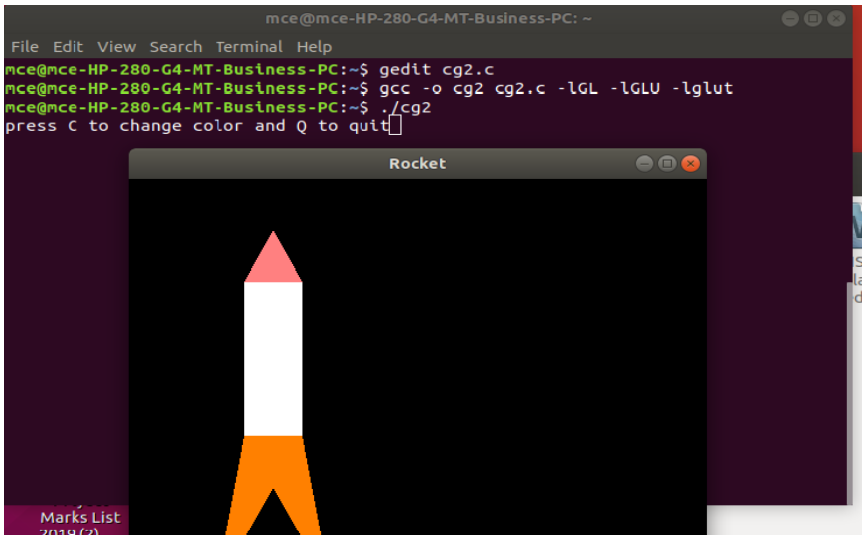
2. Write a program to draw a Rocket and allow the user to change the color.

```
#include<GL/glut.h>
#include<stdio.h>
GLfloat  r1=1.0, r2=0.5,r3=0.5;
GLfloat b1=1.0, b2=1.0, b3=1.0;
GLfloat t1=1.0, t2=0.5, t3=0.0;
void change();
void display();

void  top()
{
        glBegin(GL_TRIANGLES);
        glColor3f(r1,r2,r3);
        glVertex2f(100.0,400.0);
        glVertex2f(150.0,400.0);
        glVertex2f(125.0,450.0);
        glEnd();
        glFlush();
}

void  body()
{
        glBegin(GL_QUADS);
        glColor3f(b1,b2,b3);
        glVertex2f(100.0,400.0);
        glVertex2f(150.0,400.0);
        glVertex2f(150.0,250.0);
        glVertex2f(100.0,250.0);
        glEnd();
        glFlush();
}

void  tail()
{
        glBegin(GL_TRIANGLES);
        glColor3f(t1,t2,t3);
        glVertex2f(100.0,250.0);
        glVertex2f(150.0,250.0);
        glVertex2f(75.0,100.0);
```

```
        glVertex2f(100.0,250.0);
        glVertex2f(150.0,250.0);
        glVertex2f(175.0,100.0);
        glEnd();
        glFlush();
}

void display()
{
        char key;
        glClear(GL_COLOR_BUFFER_BIT);
        top();
        body();
        tail();
        printf("press C to change color and Q to quit");
        scanf("%c",&key);
        if(key=='c'||key=='C')
                change();
        if(key=='q'||key=='Q')
        exit(0);
}

void change()
{
        int x;
        do
        {
                printf("1.top\n 2.body\n 3.tail\n 4.quit\n enter your choice\n");
                scanf("%d",&x);
                switch(x)
                {
                        case 1:printf("enter components of new top\n");
                        scanf("%f%f%f",&r1,&r2,&r3);
                        break;
                        case 2:printf("enter components of new body\n");
                         scanf("%f%f%f",&b1,&b2,&b3);
                        break;
                        case 3:printf("enter components of new tail\n");
                        scanf("%f%f%f",&t1,&t2,&t3);
                        break;
                        case 4:break;
```

```
                }
        }while(x!=4);
display();
}

void myinit()
{
        glClearColor(0.0,0.0,0.0,1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,500.0,0.0,500.0);
        glMatrixMode(GL_MODELVIEW);
}

void main(int argc, char**argv)
{
         glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(750.0,0.0);
        glutCreateWindow("Rocket");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

Output:



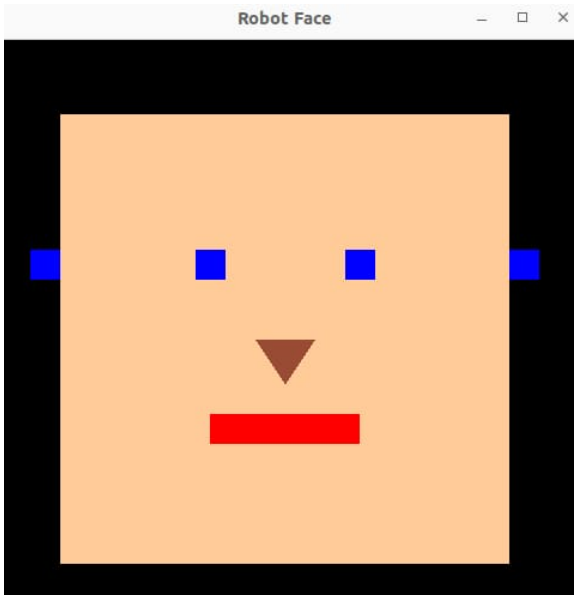
3. Write a program create robot face using display list

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

// Define display list identifiers
#define EYE 1
#define FACE 2

void createEye() {
   glNewList(EYE, GL_COMPILE);
   glBegin(GL_POLYGON);
   glColor3f(0.0, 0.0, 1.0); // Blue eye
   // Drawing a small box (rectangle) instead of a circle
   glVertex2f(-1.0, 1.0); // Top-left corner of the rectangle
   glVertex2f(1.0, 1.0);  // Top-right corner of the rectangle
   glVertex2f(1.0, -1.0); // Bottom-right corner of the rectangle
   glVertex2f(-1.0, -1.0); // Bottom-left corner of the rectangle
   glEnd();
   glEndList();
}

void createFace() {
   glNewList(FACE, GL_COMPILE);
   // Draw face outline
   glBegin(GL_POLYGON);
   glColor3f(1.0, 0.8, 0.6); // Skin color
   glVertex2f(-15.0, 15.0);
   glVertex2f(15.0, 15.0);
   glVertex2f(15.0, -15.0);
   glVertex2f(-15.0, -15.0);
   glEnd();

   // Right eye
   glPushMatrix();
   glTranslatef(-5.0, 5.0, 0.0); // Move to right position
   glCallList(EYE); // Call the eye display list
   glPopMatrix();

   // Left eye
   glPushMatrix();
   glTranslatef(5.0, 5.0, 0.0); // Move to left position
   glCallList(EYE); // Call the eye display list
   glPopMatrix();

   // Nose (small triangle)
```

```
    glBegin(GL_TRIANGLES);
    glColor3f(0.6, 0.3, 0.2); // Darker skin tone
    glVertex2f(-2.0, 0.0);
    glVertex2f(2.0, 0.0);
    glVertex2f(0.0, -3.0);
    glEnd();

    // Mouth (rectangle)
    glBegin(GL_POLYGON);
    glColor3f(1.0, 0.0, 0.0); // Red mouth
    glVertex2f(-5.0, -5.0);
    glVertex2f(5.0, -5.0);
    glVertex2f(5.0, -7.0);
    glVertex2f(-5.0, -7.0);
    glEnd();

    // Right Ear (using the EYE display list)
    glPushMatrix();
    glTranslatef(16.0, 5.0, 0.0); // Position the right ear
    glCallList(EYE); // Call the eye display list for right ear
    glPopMatrix();

    // Left Ear (using the EYE display list)
    glPushMatrix();
    glTranslatef(-16.0, 5.0, 0.0); // Position the left ear
    glCallList(EYE); // Call the eye display list for left ear
    glPopMatrix();

    glEndList();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glCallList(FACE); // Call face display list
    glFlush();
}

void myInit() {
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-20.0, 20.0, -20.0, 20.0); // Set the 2D orthographic projection
    glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char **argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
```

```
    glutInitWindowSize(500, 500);
    glutCreateWindow("Robot Face");
    myInit();
    createEye(); // Create the eye display list
    createFace(); // Create the face display list
    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

Output:

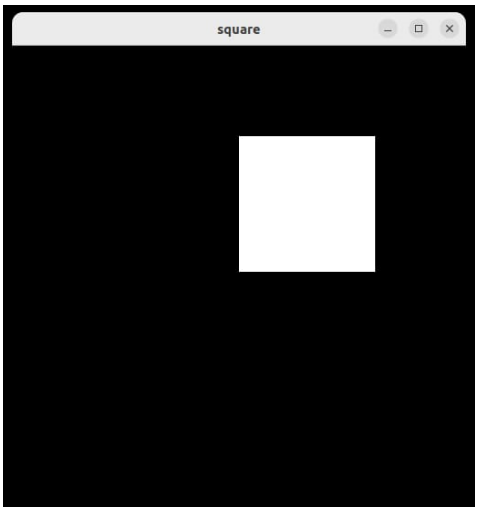


4. Write a program to generate square for right click and to exit for left click using mouse function.

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

void myDisplay()
{
        glClear(GL_COLOR_BUFFER_BIT);
}

void drawsquare()
{
        glBegin(GL_POLYGON);
        glVertex2f(0.0,30.0);
        glVertex2f(0.0,0.0);
        glVertex2f(30.0,0.0);
        glVertex2f(30.0,30.0);
        glEnd();
        glFlush();
}

void mymouse(int btb, int state, int x, int y)
{
        if(btb==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
                drawsquare();
        if(btb==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
                exit(0);
}
void myinit()
{
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-50.0, 50.0, -50.0, 50.0);
   glMatrixMode(GL_MODELVIEW);
}

int  main(int argc,char **argv)
{

        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutCreateWindow("square");
```

```
        myinit();
        glutDisplayFunc(myDisplay);
        glutMouseFunc(mymouse);
        glClearColor(0.0,0.0,0.0,0.0);
        glutMainLoop();
}
```

Output:

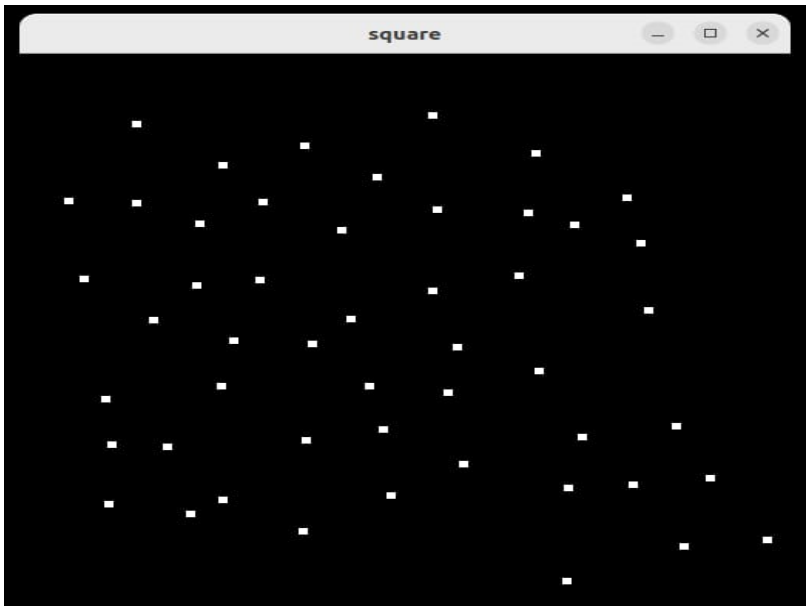


5. Write a program to draw box at each location on the screen where the mouse cursor is located.

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

GLSizei wh=500, ww=500;   //initial window width and height
Glfloat size = 3.0;

void myDisplay()
{
        glClear(GL_COLOR_BUFFER_BIT);
}

void drawsquare(int x, int y)
{
        y=wh-y;
        glBegin(GL_POLYGON);
        glVertex2f(x+size, y+size);
        glVertex2f(x-size, y+size);
        glVertex2f(x-size, y-size);
        glVertex2f(x+size, y-size);
        glEnd();
        glFlush();
}

Void mymouse(int btb, int state, int x, int y)
{
        if(btn==GLUT_LEFT_BUTTON&&state==GLUT_DOWN)
        drawsquare(x,y);
         if(btn==GLUT_RIGHT_BUTTON&&state==GLUT_DOWN)
        exit();



void  myReshape (GLsizei w, GLsizei h)
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, (GLdouble)w, 0.0, (GLdouble)h);
        glMatrixMode(GL_MODELVIEW);
        glViewport(0,0,w,h);
```

```
        ww=w;
        wh=h;
}

int  main(int argc,char **argv)
{

        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutCreateWindow("square");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(myDisplay);
         glutMouseFunc(myMouse);
        glClearColor(1.0,1.0,1.0,1.0);
        glutMainLoop();
        }
```

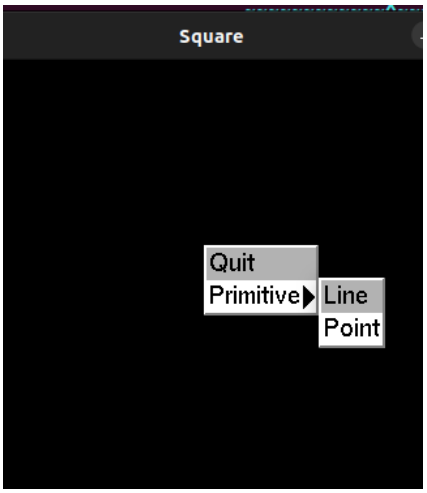
Output:



6. Write a program to create hierarchical menus.

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

void myDisplay() {
   glClear(GL_COLOR_BUFFER_BIT);
   glFlush();
}

void line() {
   glBegin(GL_LINES);
   glVertex2f(-0.5, 0.5);
   glVertex2f(0.5, -0.5);
   glEnd();
   glFlush();
}

void point() {
   glBegin(GL_POINTS);
   glVertex2f(0.0, 0.0);
   glEnd();
   glFlush();
}

void main_menu(int id) {
   switch (id) {
      case 1:
         exit(0);
         break;
      default:
         break;
   }
}

void sub_menu(int id) {
   switch (id) {
      case 1:
         line();
         break;
      case 2:
         point();
         break;
      default:
         break;
   }
}
```

```
void myinit() {
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-1.0, 1.0, -1.0, 1.0);
   glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char **argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutCreateWindow("Square");
   glutDisplayFunc(myDisplay);
   myinit();
   int subMenu = glutCreateMenu(sub_menu);
   glutAddMenuEntry("Line", 1);
   glutAddMenuEntry("Point", 2);
   glutCreateMenu(main_menu);
   glutAddMenuEntry("Quit", 1);
   glutAddSubMenu("Primitive", subMenu);
   glutAttachMenu(GLUT_RIGHT_BUTTON);
   glutMainLoop();
   return 0;
}
```



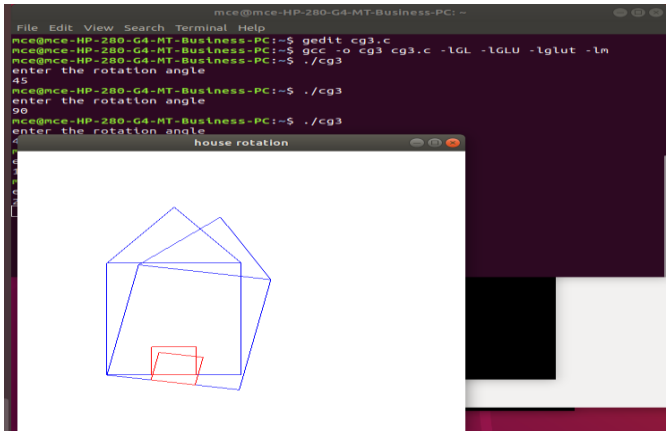
7. **Write a program to create a house like figure and rotate it about a given fixed point using OpenGL functions.**

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

GLfloat house[3][9]={ {100.0,100.0,175.0,250.0,250.0,150.0,150.0,200.0,200.0},
                      {100.0,300.0,400.0,300.0,100.0,100.0,150.0,150.0,100.0},
                       {1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 1.0 } };
GLfloat rot_mat[3][3]={{0},{0},{0}};
GLfloat result[3][9]={{0},{0},{0}};
GLfloat h=100.0;
GLfloat k=100.0,theta;
void multiply()
{
        int i,j,l;
        for(i=0;i<3;i++)
        for(j=0;j<9;j++)
            {
             result[i][j]=0;
             for(l=0;l<3;l++)
                 result[i][j]=result[i][j]+rot_mat[i][l]*house[l][j];
             }
}

void rotate()
{
        GLfloat m,n;
        m=-h*(cos(theta)-1)+k*(sin(theta));
        n=-k*(cos(theta)-1)-h*(sin(theta));
        rot_mat[0][0]=cos(theta);
        rot_mat[0][1]=-sin(theta);
        rot_mat[0][2]=m;
        rot_mat[1][0]=sin(theta);
        rot_mat[1][1]=cos(theta);
        rot_mat[1][2]=n;
        rot_mat[2][0]=0;
        rot_mat[2][1]=0;
        rot_mat[2][2]=1;
        multiply();
}

void draw_house()
{
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(house[0][0],house[1][0]);
```

```
        glVertex2f(house[0][1],house[1][1]);
        glVertex2f(house[0][3],house[1][3]);
        glVertex2f(house[0][4],house[1][4]);
        glEnd();
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(house[0][5],house[1][5]);
        glVertex2f(house[0][6],house[1][6]);
        glVertex2f(house[0][7],house[1][7]);
        glVertex2f(house[0][8],house[1][8]);
        glEnd();
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(house[0][1],house[1][1]);
        glVertex2f(house[0][2],house[1][2]);
        glVertex2f(house[0][3],house[1][3]);
        glEnd();
}

void drawrotatedhouse()
{
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(result[0][0],result[1][0]);
        glVertex2f(result[0][1],result[1][1]);
        glVertex2f(result[0][3],result[1][3]);
        glVertex2f(result[0][4],result[1][4]);
        glEnd();
        glColor3f(1.0,0.0,0.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(result[0][5],result[1][5]);
        glVertex2f(result[0][6],result[1][6]);
        glVertex2f(result[0][7],result[1][7]);
        glVertex2f(result[0][8],result[1][8]);
        glEnd();
        glColor3f(0.0,0.0,1.0);
        glBegin(GL_LINE_LOOP);
        glVertex2f(result[0][1],result[1][1]);
        glVertex2f(result[0][2],result[1][2]);
        glVertex2f(result[0][3],result[1][3]);
        glEnd();
}

void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        draw_house();
```

```
        rotate();
        drawrotatedhouse();
        glFlush();
}

void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        glPointSize(1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,500.0,0.0,500.0);
      glMatrixMode(GL_MODELVIEW);

}
int main(int argc,char *argv[])
{
        printf("enter the rotation angle\n");
        scanf("%f",&theta);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("house rotation");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

Output:

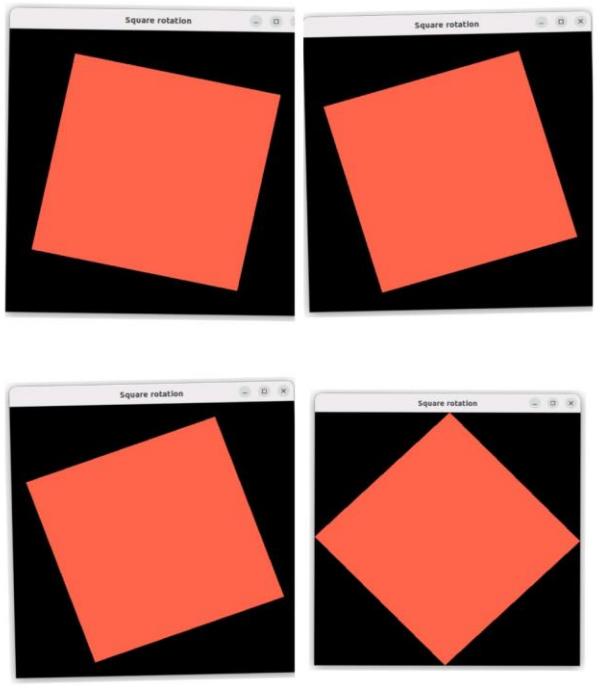


8. Write a program to create a Rotating Square.

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

GLfloat theta, thetar;

void display(){
        GLfloat thetar=theta/(3.142/180);
        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(0.0,0.0,0.0,1.0);
        glBegin(GL_POLYGON);
        glColor3f(1.0,0.4,0.3);

        glVertex2f(-cos(thetar),-sin(thetar));
        glVertex2f(sin(thetar),-cos(thetar));
        glVertex2f(cos(thetar),sin(thetar));
        glVertex2f(-sin(thetar),cos(thetar));

        glEnd();
        glFlush();
}

void idle(){
          theta += 0.0000002;
        if((theta)>= 360.0)
                theta -=0.000002;
          glutPostRedisplay();
}

void myMouse(int button,int state,int x,int y){
        if(button==GLUT_LEFT_BUTTON && state==GLUT_DOWN)
         glutIdleFunc(idle);
        if(button==GLUT_RIGHT_BUTTON && state==GLUT_DOWN)
        glutIdleFunc(NULL);
}

void mykey(unsigned char key,int x, int y){
        if (key=='q'|| key=='Q')
         exit(0);
}

void myinit()
{
        glMatrixMode(GL_PROJECTION);
        gluOrtho2D(0.0,500.0,0.0,500.0);
```

```
        glLoadIdentity();
        glMatrixMode(GL_MODELVIEW);
}

void main(int argc,char**argv){
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Rotating Square");
        glutMouseFunc(myMouse);
        glutKeyboardFunc(mykey);
        glutDisplayFunc(display);
        idle();
        myinit();
        glutMainLoop();
}
```

Output:



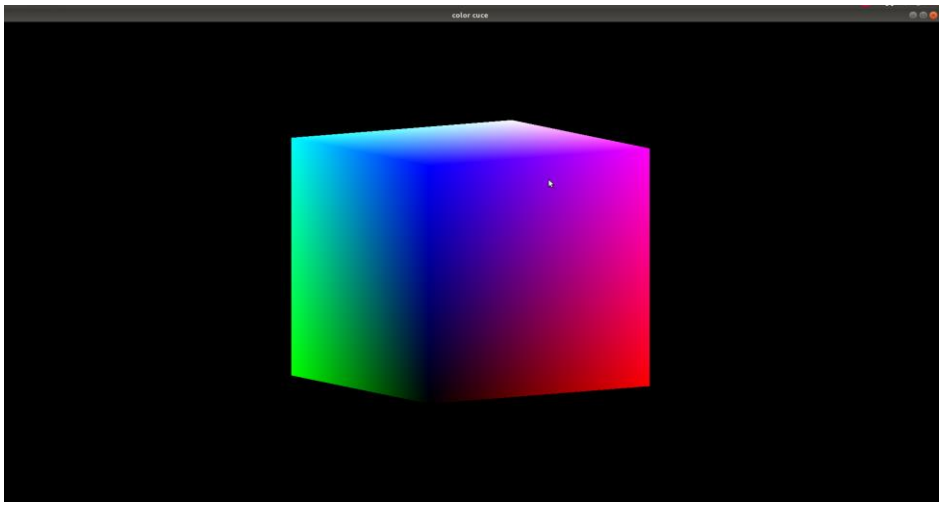
9. Write a program to draw a color cube and spin it using OpenGL transformation matrices.

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

GLfloat    vertices[ ]={ -1.0,-1.0,-1.0,
                          1.0,-1.0,-1.0,
                          1.0, 1.0,-1.0,
                        - 1.0, 1.0,-1.0,
                        - 1.0,-1.0, 1.0,
                          1.0,-1.0, 1.0,
                          1.0, 1.0, 1.0,
                         -1.0, 1.0, 1.0 };

GLfloat    normals[ ]={ -1.0,-1.0,-1.0,
                         1.0,-1.0,-1.0,
                         1.0, 1.0,-1.0,
                        -1.0, 1.0,-1.0,
                        -1.0,-1.0, 1.0,
                         1.0,-1.0, 1.0,
                         1.0, 1.0, 1.0,
                        -1.0, 1.0, 1.0 };

GLfloat      colors[ ]={  0.0,0.0,0.0,
                          1.0,0.0,0.0,
                          1.0,1.0,0.0,
                          0.0,1.0,0.0,
                          0.0,0.0,1.0,
                          1.0,0.0,1.0,
                          1.0,1.0,1.0,
                          0.0,1.0,1.0};

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

static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;

void display(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);
        glFlush();
        glutSwapBuffers();
}

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

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

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);
}


int main(int argc,char **argv)
{
                glutInit(&argc,argv);
                glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
                glutInitWindowSize(500,500);
                glutCreateWindow("color cuce");
                glutReshapeFunc(myReshape);
                glutDisplayFunc(display);
                glutMouseFunc(mouse);
                 glutIdleFunc(spincube);
                glEnable(GL_DEPTH_TEST);
                glEnableClientState(GL_COLOR_ARRAY);
```

```
        glEnableClientState(GL_VERTEX_ARRAY);
        glEnableClientState(GL_NORMAL_ARRAY);
        glVertexPointer(3,GL_FLOAT,0,vertices);
        glColorPointer(3,GL_FLOAT,0,colors);
        glNormalPointer(GL_FLOAT,0,normals);
        glColor3f(1.0,1.0,1.0);
        glutMainLoop();
}
```

Output



10. lip the line segment A(-4,2) and B(-1,7) in a window defined by left bottom corner at (-3,1) and upper right corner at (2,6). Find the visible portion of the line segment using Cohen Sutherland line clipping algorithm.

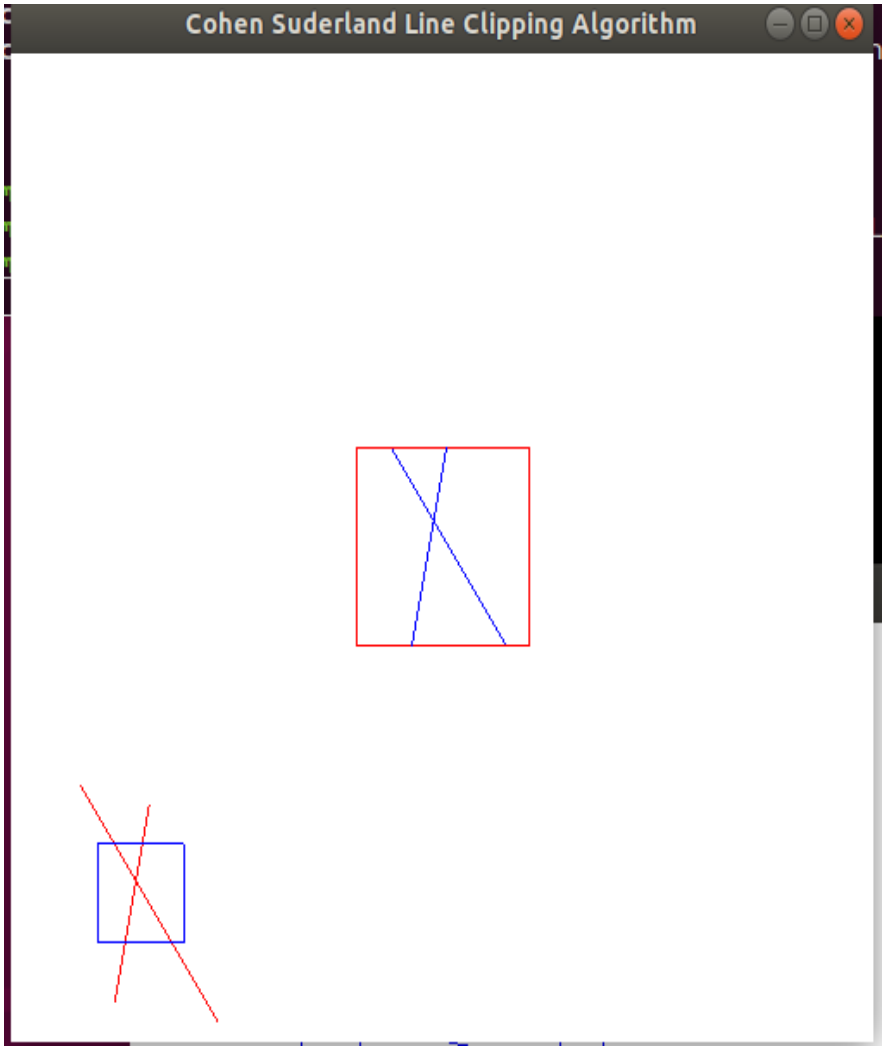
```
#include <stdio.h>
#include <GL/glut.h>
#define outcode int
#define true 1
#define false 0

double xmin=50,ymin=50, xmax=100,ymax=100; // Window boundaries
double xvmin=200,yvmin=200,xvmax=300,yvmax=300; // Viewport boundaries
//int x1, x2, y1, y2;
//bit codes for the right, left, top, & bottom
const int RIGHT = 8;
const int LEFT = 2;
const int TOP = 4;
const int BOTTOM = 1;

//used to compute bit codes of a point
outcode ComputeOutCode (double x, double y);

//Cohen-Sutherland clipping algorithm clips a line from
//P0 = (x0, y0) to P1 = (x1, y1) against a rectangle with
//diagonal from (xmin, ymin) to (xmax, ymax).
void CohenSutherlandLineClipAndDraw (double x0, double y0,double x1, double y1)
{
        //Outcodes for P0, P1, and whatever point lies outside the clip rectangle
        outcode outcode0, outcode1, outcodeOut;
        bool accept = false, done = false;

        //compute outcodes
        outcode0 = ComputeOutCode (x0, y0);
        outcode1 = ComputeOutCode (x1, y1);

        do{
                if (!(outcode0 | outcode1))      //logical or is 0 Trivially accept & exit
                {
                        accept = true;
                        done = true;
                }
                else if (outcode0 & outcode1)   //logical and is not 0. Trivially reject and exit
                        done = true;
                else
                {
                        //failed both tests, so calculate the line segment to clip
                        //from an outside point to an intersection with clip edge
```

```
                double x, y;

                //At least one endpoint is outside the clip rectangle; pick it.
                outcodeOut = outcode0? outcode0: outcode1;

                //Now find the intersection point;
                //use formulas y = y0 + slope * (x - x0), x = x0 + (1/slope)* (y - y0)
                if (outcodeOut & TOP)           //point is above the clip rectangle
                {
                        x = x0 + (x1 - x0) * (ymax - y0)/(y1 - y0);
                        y = ymax;
                }
                else if (outcodeOut & BOTTOM)  //point is below the clip rectangle
                {
                        x = x0 + (x1 - x0) * (ymin - y0)/(y1 - y0);
                        y = ymin;
                }
                else if (outcodeOut & RIGHT)   //point is to the right of clip rectangle
                {
                        y = y0 + (y1 - y0) * (xmax - x0)/(x1 - x0);
                        x = xmax;
                }
                else                    //point is to the left of clip rectangle
                {
                        y = y0 + (y1 - y0) * (xmin - x0)/(x1 - x0);
                        x = xmin;
                }

                //Now we move outside point to intersection point to clip
                //and get ready for next pass.
                if (outcodeOut == outcode0)
                {
                        x0 = x;
                        y0 = y;
                        outcode0 = ComputeOutCode (x0, y0);
                }
                else
                {
                        x1 = x;
                        y1 = y;
                        outcode1 = ComputeOutCode (x1, y1);
                }
        }
}while (!done);

if (accept)
{               // 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;
                        //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();
                glColor3f(0.0,0.0,1.0); // draw blue colored clipped line
                glBegin(GL_LINES);
                        glVertex2d (vx0, vy0);
                        glVertex2d (vx1, vy1);
                glEnd();
        }
}

//Compute the bit code for a point (x, y) using the clip rectangle
//bounded diagonally by (xmin, ymin), and (xmax, ymax)
outcode ComputeOutCode (double x, double y)
{
        outcode code = 0;
        if (y > ymax)           //above the clip window
                code |= TOP;
        else if (y < ymin)       //below the clip window
                code |= BOTTOM;
        if (x > xmax)           //to the right of clip window
                code |= RIGHT;
        else if (x < xmin)       //to the left of clip window
                code |= LEFT;
        return code;
}

void display()
{
double x0=120,y0=10,x1=40,y1=130;
glClear(GL_COLOR_BUFFER_BIT);
//draw the line with red color
glColor3f(1.0,0.0,0.0);
//bres(120,20,340,250);
glBegin(GL_LINES);
                        glVertex2d (x0, y0);
```

```
                        glVertex2d (x1, y1);
                        glVertex2d (60,20);
                        glVertex2d (80,120);
                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();
CohenSutherlandLineClipAndDraw(x0,y0,x1,y1);
CohenSutherlandLineClipAndDraw(60,20,80,120);
glFlush();
}
void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        glPointSize(1.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0,499.0,0.0,499.0);
}
int main(int argc, char** argv)
{

        //printf("Enter End points:");
        //scanf("%d%d%d%d", &x1,&x2,&y1,&y2);

        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Cohen Suderland Line Clipping Algorithm");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

Output:


Cohen Suderland Line Clipping Algorithm