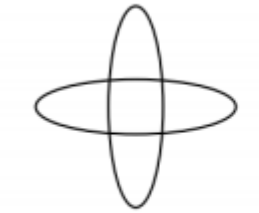
CG Lab 6

SAHIL BONDRE: U18CO021

1. Write a program to draw the following figure:-



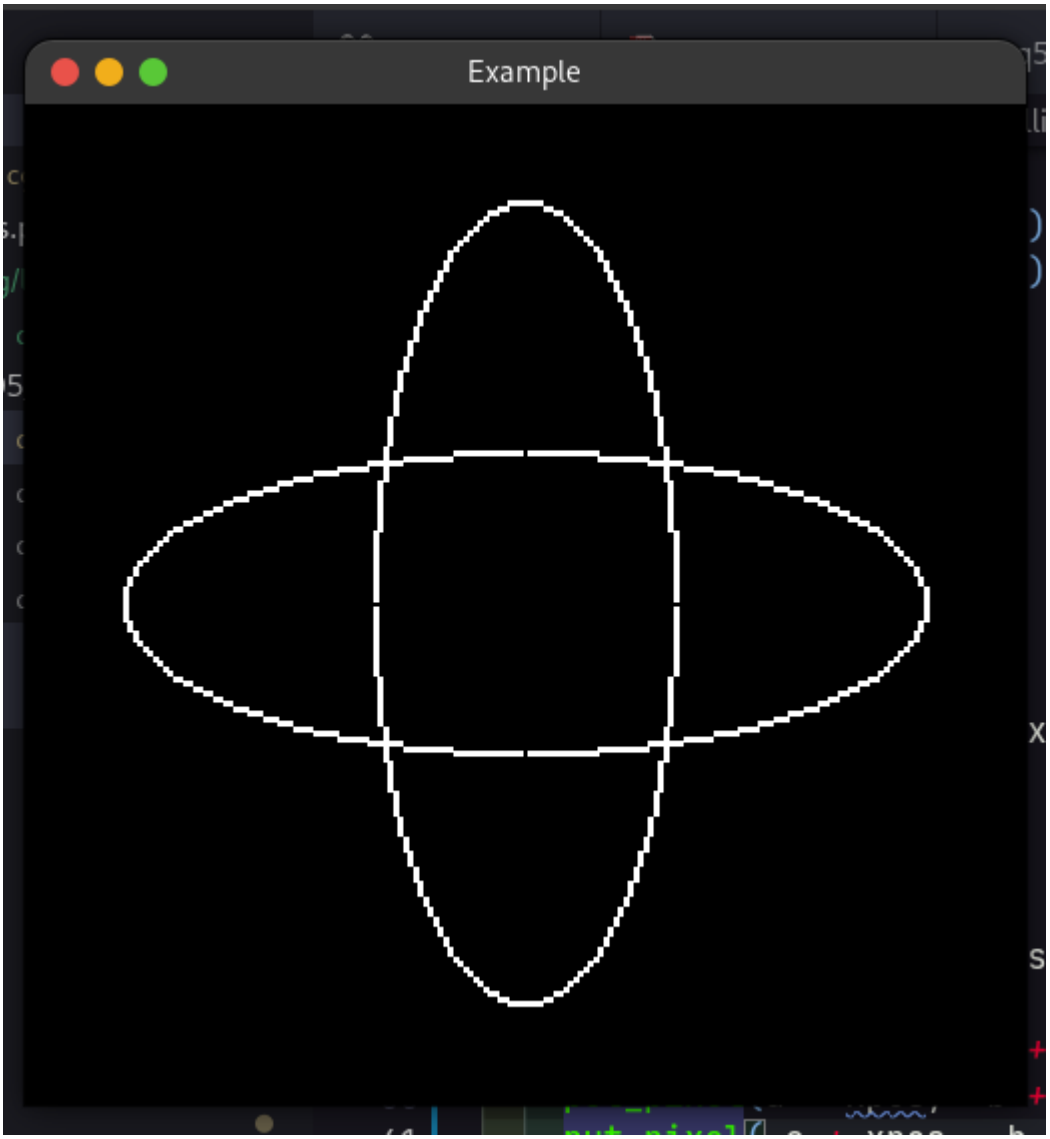
Input is r_x, r_y and center coordinates.

(Use the concept of Mid Point Ellipse generating algorithm)

```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>

using namespace std;

int rx = 0, ry = 0, x = 0, y = 0;

void init(void) { glClearColor(0, 0, 0, 0); }

void put_pixel(int x, int y) { glVertex2d(x, y); }

void draw_ellipse() {
  vector<pair<int, int>> coords, temp;
  int x = 0, y = ry;
  int p = ry * ry - rx * rx * ry + rx * rx / 4;
  int dx = 2 * ry * ry * x;
  int dy = 2 * rx * rx * y;

  while (dx < dy) {
    coords.push_back({x, y});
    coords.push_back({y, x});
    if (p < 0) {
      ++x;
      dx = 2 * ry * ry * x;
      p += 2 * ry * ry * x + ry * ry;
    } else {
```

```
      ++x;
      --y;
      dx = 2 * ry * ry * x;
      dy = 2 * rx * rx * y;
      p += (2 * ry * ry * x) + (ry * ry) - (2 * rx * rx * y);
    }
  }

  p = ry * ry * (x + 0.5) * (x + 0.5) + rx * rx * (y - 1) * (y - 1) -
      rx * rx * ry * ry;

  while (y > 0) {
    coords.push_back({x, y});
    coords.push_back({y, x});
    if (p > 0) {
      --y;
      dy = 2 * rx * rx * y;
      p += rx * rx - dy;
    } else {
      ++x;
      --y;
      dx = 2 * ry * ry * x;
      dy = 2 * rx * rx * y;
      p += dx - dy + rx * rx;
    }
  }

  for (auto p : coords) {
    int a = p.first, b = p.second;
    put_pixel(a + x, b + y);
    put_pixel(-a + x, b + y);
    put_pixel(a + x, -b + y);
    put_pixel(-a + x, -b + y);
  }
}

void display() {

  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  glPointSize(3.0);

  glBegin(GL_POINTS);

  draw_ellipse();
```

```
  glEnd();

  glFlush();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluOrtho2D(-100, 100, -100, 100);

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

int main(int argc, char **argv) {
  cout << "Enter radii of ellipse: ";
  cin >> rx >> ry;
  cout << "Enter x and y: ";
  cin >> x >> y;
  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB);

  glutCreateWindow("Example");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
}
```



2. Write a menu driven Program to implement a set of basic Transformations on Polygon. Program should include: Translation, Rotation and Scaling.

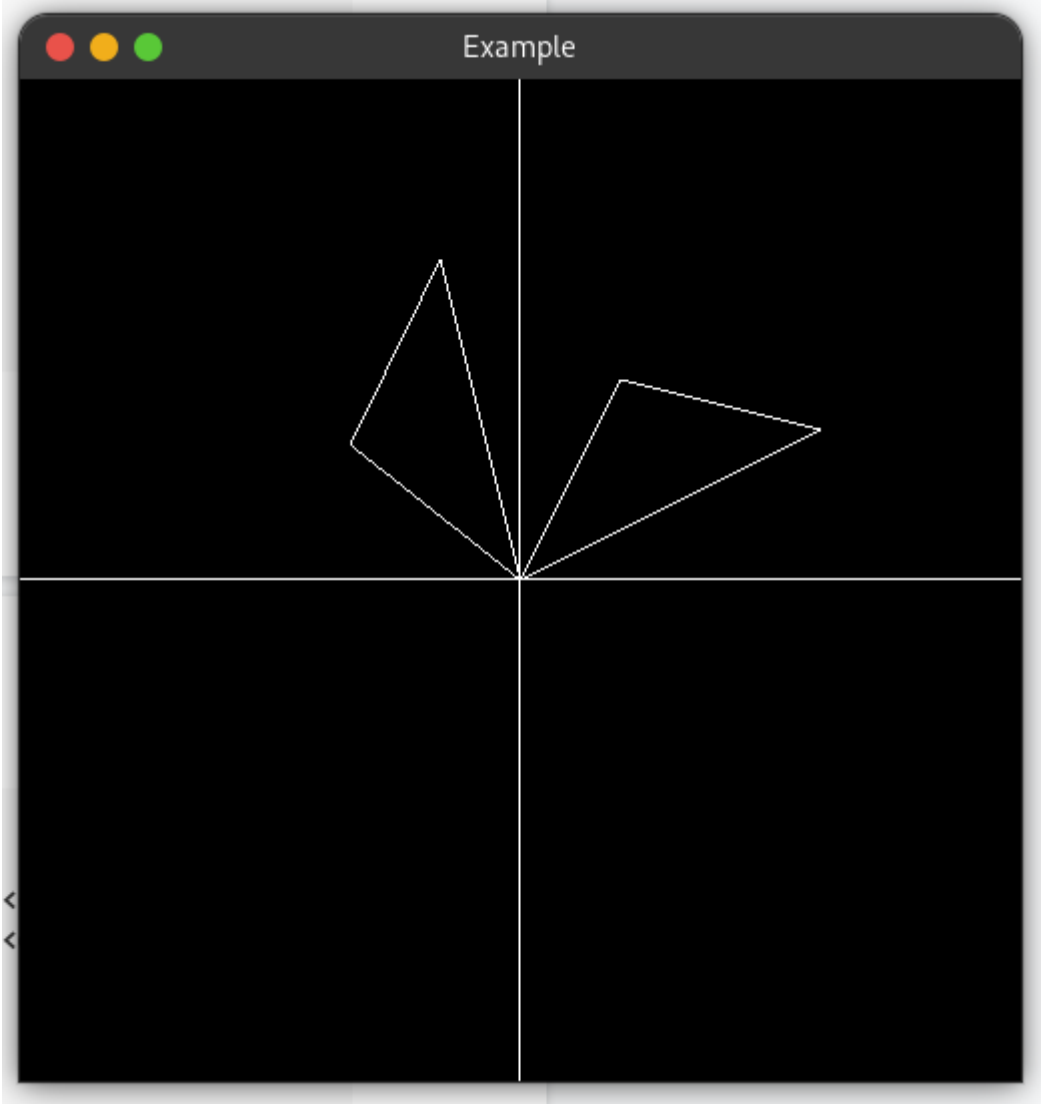
```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>

using namespace std;

vector<vector<double>> points;
vector<vector<double>> transformation_matrix = {
    {1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

void init(void) { glClearColor(0, 0, 0, 0); }
```

```
void put_pixel(int x, int y) { glVertex2d(x, y); }

vector<vector<double>> matrix_multiply(vector<vector<double>> a,
                                       vector<vector<double>> b) {
  int r1 = a.size();
  int c1 = a[0].size();
  int r2 = b.size();
  int c2 = b[0].size();

  if (c1 != r2) {
    cout << "Error in matrix sizes: " << r1 << " " << c1 << " " << r2 <<
" "
         << c2 << "\n";
    exit(EXIT_FAILURE);
  }

  vector<vector<double>> mul(r1, vector<double>(c2));

  for (int i = 0; i < r1; i++) {
    for (int j = 0; j < c2; j++) {
      mul[i][j] = 0;
      for (int k = 0; k < c1; k++) {
        mul[i][j] += a[i][k] * b[k][j];
      }
    }
  }

  return mul;
}

void translation(double x, double y) {
  vector<vector<double>> v = {{1, 0, 0}, {0, 1, 0}, {x, y, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void rotation(double deg) {
  double cosVal = cos(deg * M_PI / 180);
  double sinVal = sin(deg * M_PI / 180);

  vector<vector<double>> v = {
      {cosVal, sinVal, 0}, {-sinVal, cosVal, 0}, {0, 0, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void scaling(double sx, double sy) {
  vector<vector<double>> v = {{sx, 0, 0}, {0, sy, 0}, {0, 0, 1}};
```

```
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void draw_points() {
  glBegin(GL_LINE_LOOP);

  for (auto p : points) {
    put_pixel(p[0], p[1]);
  }

  glEnd();
}

void draw_axis() {
  glBegin(GL_LINES);

  put_pixel(-100, 0);
  put_pixel(100, 0);
  put_pixel(0, 100);
  put_pixel(0, -100);

  glEnd();
}

void display() {

  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  glPointSize(3.0);
  draw_axis();
  draw_points();
  points = matrix_multiply(points, transformation_matrix);
  draw_points();

  glFlush();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluOrtho2D(-100, 100, -100, 100);

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

```
}

int main(int argc, char **argv) {
  cout << "Enter number of coordinate points: ";
  points = vector<vector<double>>(0);
  int n;
  cin >> n;
  int x, y;
  for (int i = 0; i < n; ++i) {
    cout << "Enter points " << i + 1 << ": ";
    cin >> x >> y;
    points.push_back({(double)x, (double)y, 1.0});
  }

  cout << "Enter s for scaling, t for translation, r for rotation: ";
  string s;
  cin >> s;

  if (s == "s") {
    double sx, sy;
    cout << "Enter sx, sy: ";
    cin >> sx >> sy;
    scaling(sx, sy);

  } else if (s == "t") {
    double tx, ty;
    cout << "Enter tx, ty: ";
    cin >> tx >> ty;
    translation(tx, ty);

  } else if (s == "r") {
    double degree;
    cout << "Enter degree: ";
    cin >> degree;
    rotation(degree);
  }

  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB);

  glutCreateWindow("Example");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
```

```
  glutMainLoop();
}
```



3. Write a menu driven program to implement set of Composite Transformations on Polygon: Program should include: Translation, Rotation (about arbitrary point, arbitrary axis, and arbitrary plane), Scaling (fixed point), and Shearing (X & Y), Reflection (along X axis, along y axis, along the origin and along Y=X line).

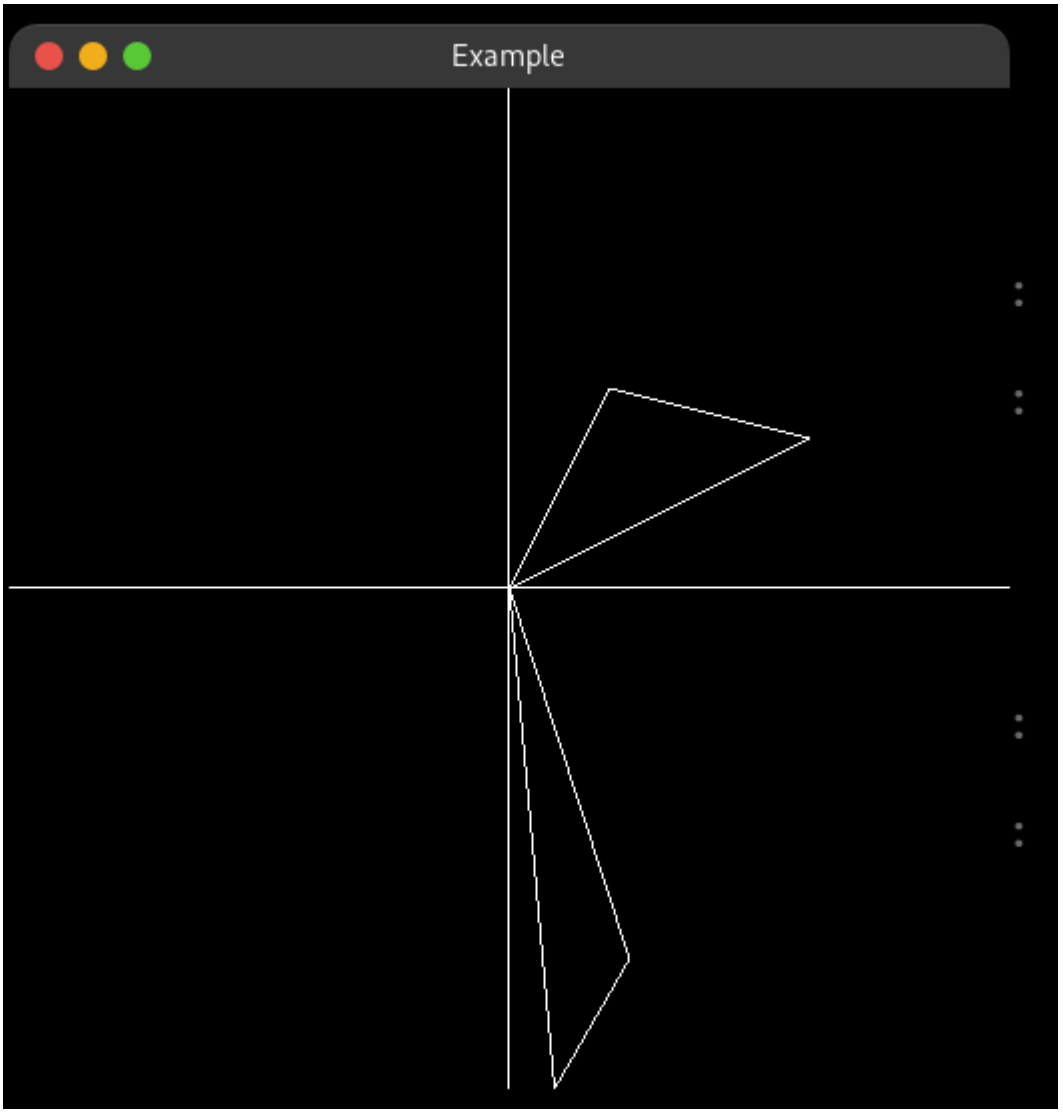
```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>

using namespace std;

vector<vector<double>> points;
vector<vector<double>> transformation_matrix = {
    {1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

void init(void) { glClearColor(0, 0, 0, 0); }

void put_pixel(int x, int y) { glVertex2d(x, y); }

vector<vector<double>> matrix_multiply(vector<vector<double>> a,
                                       vector<vector<double>> b) {
  int r1 = a.size();
  int c1 = a[0].size();
  int r2 = b.size();
  int c2 = b[0].size();

  if (c1 != r2) {
    cout << "Error in matrix sizes: " << r1 << " " << c1 << " " << r2 <<
" "
         << c2 << "\n";
    exit(EXIT_FAILURE);
  }

  vector<vector<double>> mul(r1, vector<double>(c2));

  for (int i = 0; i < r1; i++) {
    for (int j = 0; j < c2; j++) {
      mul[i][j] = 0;
      for (int k = 0; k < c1; k++) {
        mul[i][j] += a[i][k] * b[k][j];
      }
    }
  }
```

```
  return mul;
}

void translation(double x, double y) {
  vector<vector<double>> v = {{1, 0, 0}, {0, 1, 0}, {x, y, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void rotation(double deg) {
  double cosVal = cos(deg * M_PI / 180);
  double sinVal = sin(deg * M_PI / 180);

  vector<vector<double>> v = {
      {cosVal, sinVal, 0}, {-sinVal, cosVal, 0}, {0, 0, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void scaling(double sx, double sy) {
  vector<vector<double>> v = {{sx, 0, 0}, {0, sy, 0}, {0, 0, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void shearx(double sx) {
  vector<vector<double>> v = {{1, 0, 0}, {sx, 1, 0}, {0, 0, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void sheary(double sy) {
  vector<vector<double>> v = {{1, sy, 0}, {0, 1, 0}, {0, 0, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void refx() {
  vector<vector<double>> v = {{1, 0, 0}, {0, -1, 0}, {0, 0, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void refy() {
  vector<vector<double>> v = {{-1, 0, 0}, {0, 1, 0}, {0, 0, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void refo() {
  vector<vector<double>> v = {{-1, 0, 0}, {0, -1, 0}, {0, 0, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}
```

```
void refl() {
  vector<vector<double>> v = {{0, 1, 0}, {1, 0, 0}, {0, 0, 1}};
  transformation_matrix = matrix_multiply(transformation_matrix, v);
}

void draw_points() {
  glBegin(GL_LINE_LOOP);

  for (auto p : points) {
    put_pixel(p[0], p[1]);
  }

  glEnd();
}

void draw_axis() {
  glBegin(GL_LINES);

  put_pixel(-100, 0);
  put_pixel(100, 0);
  put_pixel(0, 100);
  put_pixel(0, -100);

  glEnd();
}

void display() {

  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  glPointSize(3.0);
  draw_axis();
  draw_points();
  points = matrix_multiply(points, transformation_matrix);
  draw_points();

  glFlush();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluOrtho2D(-100, 100, -100, 100);
```

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

int main(int argc, char **argv) {
  cout << "Enter number of coordinate points: ";
  points = vector<vector<double>>(0);
  int n;
  cin >> n;
  int x, y;
  for (int i = 0; i < n; ++i) {
    cout << "Enter points " << i + 1 << ": ";
    cin >> x >> y;
    points.push_back({(double)x, (double)y, 1.0});
  }

  while (true) {

    cout << "Enter s for scaling, t for translation, r for rotation,
s(x, y) "
            "for "
            "shearing, r(x, y, o, l) for reflection: ";
    string s;
    cin >> s;

    if (s == "s") {
      double sx, sy;
      cout << "Enter sx, sy: ";
      cin >> sx >> sy;
      scaling(sx, sy);

    } else if (s == "t") {
      double tx, ty;
      cout << "Enter tx, ty: ";
      cin >> tx >> ty;
      translation(tx, ty);

    } else if (s == "r") {
      double degree;
      cout << "Enter degree: ";
      cin >> degree;
      rotation(degree);
    } else if (s == "sx") {
      double sx;
      cout << "Enter sx: ";
```

```
      cin >> sx;
      shearx(sx);
    } else if (s == "sy") {
      double sy;
      cout << "Enter sy: ";
      cin >> sy;
      sheary(sy);
    } else if (s == "rx") {
      refx();
    } else if (s == "ry") {
      refy();
    } else if (s == "ro") {
      refo();
    } else if (s == "rl") {
      refl();
    } else {
      break;
    }
  }

  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB);

  glutCreateWindow("Example");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
}
```



4. Write a program to draw the following structure:

```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>
#include <unordered_map>
#include <vector>

using namespace std;

#define PI 3.14159265
double xLeftBoundary = -70;
double yBottomBoundary = -30;
double xRightBoundary = 70;
double upperBottomY = -1;
double xLeftUpper = -1, xRightUpper = -1;

void put_pixel(double x, double y) { glVertex2d(x, y); }

void createTriangle(double leftX, double rightX, double yCoord, double
mid) {
  glBegin(GL_LINE_LOOP);
  put_pixel(leftX - 5, yCoord);
  put_pixel(leftX + mid, yCoord + 15);
  put_pixel(rightX + 5, yCoord);
  put_pixel(rightX + 8, yCoord + 5);
  put_pixel(rightX - mid, yCoord + 23);
  put_pixel(leftX - 8, yCoord + 5);
  glEnd();
}

void slantUpperFoundation(vector<double> prevCoordLeft,
                          vector<double> prevCoordRight) {
  put_pixel(prevCoordLeft[0], prevCoordLeft[1]);
  put_pixel(prevCoordLeft[0] + 5, prevCoordLeft[1] + 5);
  put_pixel(prevCoordRight[0] - 5, prevCoordRight[1] + 5);
  put_pixel(prevCoordRight[0], prevCoordRight[1]);
}

void createSlantFoundation() {
  glBegin(GL_LINE_LOOP);
  slantUpperFoundation({xLeftBoundary, yBottomBoundary + 5 + 40},
                       {xRightBoundary, yBottomBoundary + 5 + 40});
  glEnd();

  double xLeft1 = xLeftBoundary + 5;
  double xRight1 = xRightBoundary - 5;
```

```
  double y1 = yBottomBoundary + 45 + 5;

  glBegin(GL_LINE_LOOP);
  slantUpperFoundation({xLeft1, y1}, {xRight1, y1});
  glEnd();

  double xLeft2 = xLeft1 + 5;
  double xRight2 = xRight1 - 5;
  double y2 = y1 + 5;

  glBegin(GL_LINE_LOOP);
  slantUpperFoundation({xLeft2, y2}, {xRight2, y2});
  glEnd();

  double xLeft3 = xLeft2 + 5;
  double xRight3 = xRight2 - 5;
  double y3 = y2 + 5;

  glBegin(GL_LINE_LOOP);
  slantUpperFoundation({xLeft3, y3}, {xRight3, y3});
  glEnd();

  upperBottomY = y3 + 5;
  xLeftUpper = xLeft3 + 5;
  xRightUpper = xRight3 - 5;
}

void verticalBaseFoundation(double distance) {
  put_pixel(xLeftBoundary + distance, yBottomBoundary + 10);
  put_pixel(xLeftBoundary + distance, yBottomBoundary + 10 + 30);
  put_pixel(xLeftBoundary + 5 + distance, yBottomBoundary + 10 + 30);
  put_pixel(xLeftBoundary + 5 + distance, yBottomBoundary + 10);
}

void drawOutlineFoundation() {
  glBegin(GL_LINE_LOOP);
  put_pixel(xLeftBoundary, yBottomBoundary);
  put_pixel(xLeftBoundary, yBottomBoundary + 10);
  put_pixel(xRightBoundary, yBottomBoundary + 10);
  put_pixel(xRightBoundary, yBottomBoundary);
  glEnd();

  glBegin(GL_LINE_LOOP);
  verticalBaseFoundation(5);
  verticalBaseFoundation(30);
  verticalBaseFoundation(xRightBoundary - xLeftBoundary - 35);
```

```
verticalBaseFoundation(xRightBoundary - xLeftBoundary - 10);
glEnd();

glBegin(GL_LINE_LOOP);
put_pixel(xLeftBoundary, yBottomBoundary + 40);
put_pixel(xLeftBoundary, yBottomBoundary + 5 + 40);
put_pixel(xRightBoundary, yBottomBoundary + 5 + 40);
put_pixel(xRightBoundary, yBottomBoundary + 40);
glEnd();

createSlantFoundation();

// left upper box
glBegin(GL_LINE_LOOP);
put_pixel(xLeftUpper, upperBottomY);
put_pixel(xLeftUpper, upperBottomY + 20);
put_pixel(xLeftUpper + 30, upperBottomY + 20);
put_pixel(xLeftUpper + 30, upperBottomY);
glEnd();

glBegin(GL_LINE_LOOP);
put_pixel(xLeftUpper - 5, upperBottomY + 20);
put_pixel(xLeftUpper - 5, upperBottomY + 20 + 5);
put_pixel(xLeftUpper + 30, upperBottomY + 20 + 5);
put_pixel(xLeftUpper + 30, upperBottomY + 20);
glEnd();

// right upper box
glBegin(GL_LINE_LOOP);
put_pixel(xRightUpper, upperBottomY);
put_pixel(xRightUpper, upperBottomY + 20);
put_pixel(xRightUpper - 30, upperBottomY + 20);
put_pixel(xRightUpper - 30, upperBottomY);
glEnd();

glBegin(GL_LINE_LOOP);
put_pixel(xRightUpper + 5, upperBottomY + 20);
put_pixel(xRightUpper + 5, upperBottomY + 20 + 5);
put_pixel(xRightUpper - 30, upperBottomY + 20 + 5);
put_pixel(xRightUpper - 30, upperBottomY + 20);
glEnd();

glBegin(GL_LINE_STRIP);
put_pixel(xLeftUpper - 5, upperBottomY + 25);
put_pixel(xLeftUpper + 7, upperBottomY + 25 + 15);
put_pixel(xLeftUpper + 7 + 30, upperBottomY + 25 + 15);
```

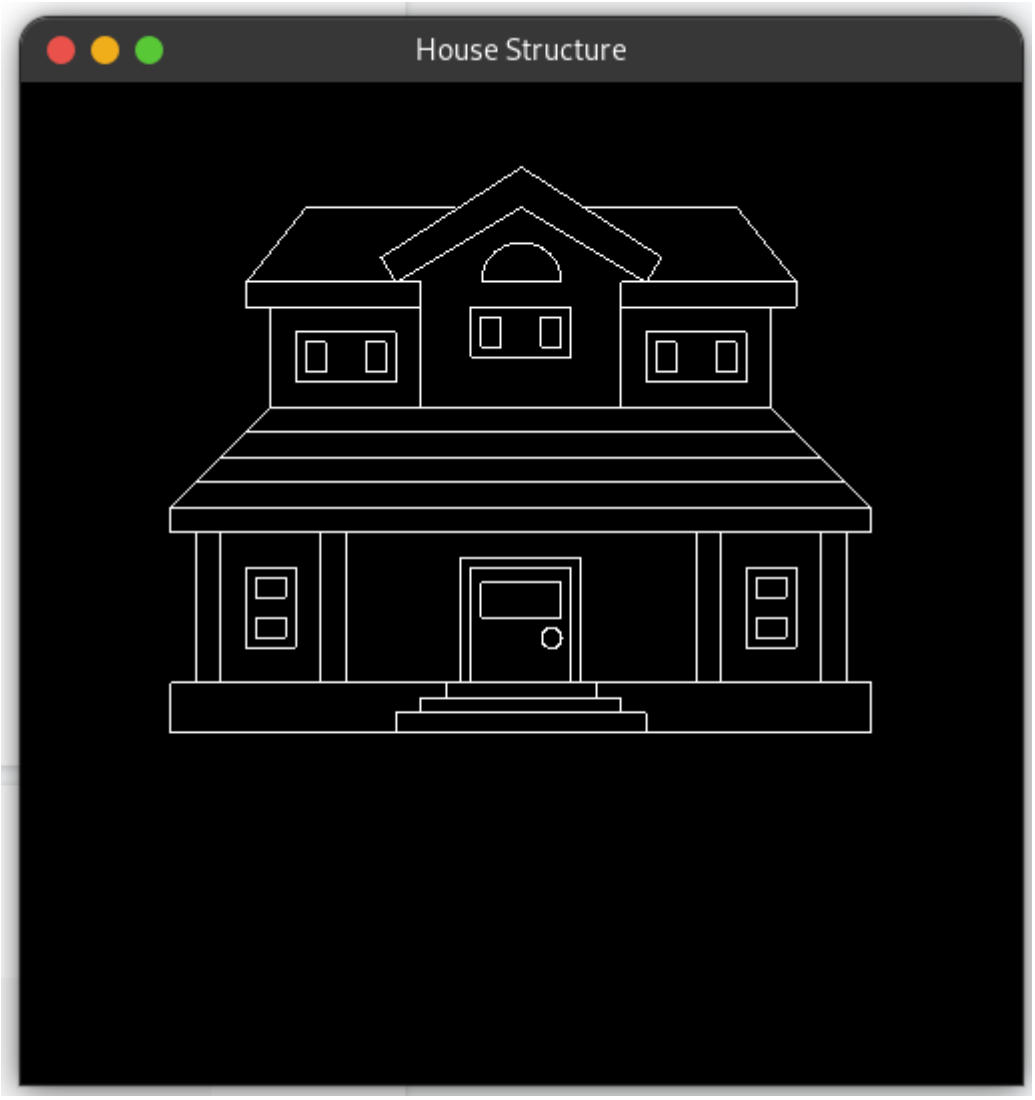
```
  glEnd();

  glBegin(GL_LINE_STRIP);
  put_pixel(xRightUpper + 5, upperBottomY + 25);
  put_pixel(xRightUpper - 7, upperBottomY + 25 + 15);
  put_pixel(xRightUpper - 7 - 30, upperBottomY + 25 + 15);
  glEnd();

  double mid = ((xRightUpper - 30) - (xLeftUpper + 30)) / 2;
  createTriangle(xLeftUpper + 30, xRightUpper - 30, upperBottomY + 25,
mid);
}

void verticalWindows(double xMiddle, double yMiddle) {
  glBegin(GL_LINE_LOOP);
  put_pixel(xMiddle - 5, yMiddle - 8);
  put_pixel(xMiddle - 5, yMiddle + 8);
  put_pixel(xMiddle + 5, yMiddle + 8);
  put_pixel(xMiddle + 5, yMiddle - 8);
  glEnd();

  glBegin(GL_LINE_LOOP);
  put_pixel(xMiddle - 3, yMiddle + 2);
  put_pixel(xMiddle - 3, yMiddle + 6);
  put_pixel(xMiddle + 3, yMiddle + 6);
  put_pixel(xMiddle + 3, yMiddle + 2);
  glEnd();

  glBegin(GL_LINE_LOOP);
  put_pixel(xMiddle - 3, yMiddle - 2);
  put_pixel(xMiddle - 3, yMiddle - 6);
  put_pixel(xMiddle + 3, yMiddle - 6);
  put_pixel(xMiddle + 3, yMiddle - 2);
  glEnd();
}

void horizontalWindows(double xMiddle, double yMiddle) {
  glBegin(GL_LINE_LOOP);
  put_pixel(xMiddle - 10, yMiddle - 5);
  put_pixel(xMiddle - 10, yMiddle + 5);
  put_pixel(xMiddle + 10, yMiddle + 5);
  put_pixel(xMiddle + 10, yMiddle - 5);
  glEnd();

  glBegin(GL_LINE_LOOP);
  put_pixel(xMiddle - 4, yMiddle - 3);
```

```
  put_pixel(xMiddle - 8, yMiddle - 3);
  put_pixel(xMiddle - 8, yMiddle + 3);
  put_pixel(xMiddle - 4, yMiddle + 3);
  glEnd();

  glBegin(GL_LINE_LOOP);
  put_pixel(xMiddle + 4, yMiddle + 3);
  put_pixel(xMiddle + 8, yMiddle + 3);
  put_pixel(xMiddle + 8, yMiddle - 3);
  put_pixel(xMiddle + 4, yMiddle - 3);
  glEnd();
}

void drawDoor(double xMiddle) {
  double yCoord = yBottomBoundary + 10;
  glBegin(GL_LINE_LOOP);
  put_pixel(xMiddle - 12, yCoord);
  put_pixel(xMiddle - 12, yCoord + 25);
  put_pixel(xMiddle + 12, yCoord + 25);
  put_pixel(xMiddle + 12, yCoord);
  glEnd();

  glBegin(GL_LINE_LOOP);
  put_pixel(xMiddle - 10, yCoord);
  put_pixel(xMiddle - 10, yCoord + 23);
  put_pixel(xMiddle + 10, yCoord + 23);
  put_pixel(xMiddle + 10, yCoord);
  glEnd();

  glBegin(GL_LINE_LOOP);
  put_pixel(xMiddle - 8, yCoord + 23 - 10);
  put_pixel(xMiddle - 8, yCoord + 23 - 3);
  put_pixel(xMiddle + 8, yCoord + 23 - 3);
  put_pixel(xMiddle + 8, yCoord + 23 - 10);
  glEnd();
}

void drawCircle(double xMiddle) {
  double yCoord = yBottomBoundary + 10;
  double yMiddle = yCoord + 23 - 10 - 4;
  float radius = 2;
  float pi = PI;
  glPointSize(1.0);
  glBegin(GL_POINTS);

  for (float i = 0.0; i <= 2 * pi; i += 0.05)
```

```
    put_pixel(xMiddle + (sin(i) * radius), yMiddle + (cos(i) * radius));

  glEnd();
}

void drawSemiCircle(double xMiddle, double yMiddle) {
  float radius = 8;
  float pi = PI;
  glPointSize(1.0);
  glBegin(GL_POINTS);

  for (float i = -1 * pi / 2; i <= pi / 2; i += 0.05)
    put_pixel(xMiddle + (sin(i) * radius), yMiddle + (cos(i) * radius));

  glEnd();

  glBegin(GL_LINE_STRIP);
  put_pixel(xMiddle + (sin(-1 * pi / 2) * radius),
            yMiddle + (cos(-1 * pi / 2) * radius));
  put_pixel(xMiddle + (sin(pi / 2) * radius), yMiddle + (cos(pi / 2) *
radius));
  glEnd();
}

void drawInnerDesigns() {
  double mid = (xRightBoundary + xLeftBoundary) / 2;
  glBegin(GL_LINE_LOOP);
  put_pixel(mid - 25, yBottomBoundary);
  put_pixel(mid - 25, yBottomBoundary + 4);
  put_pixel(mid + 25, yBottomBoundary + 4);
  put_pixel(mid + 25, yBottomBoundary);
  glEnd();

  glBegin(GL_LINE_LOOP);
  put_pixel(mid - 20, yBottomBoundary + 4);
  put_pixel(mid - 20, yBottomBoundary + 7);
  put_pixel(mid + 20, yBottomBoundary + 7);
  put_pixel(mid + 20, yBottomBoundary + 4);
  glEnd();

  glBegin(GL_LINE_LOOP);
  put_pixel(mid - 15, yBottomBoundary + 7);
  put_pixel(mid - 15, yBottomBoundary + 10);
  put_pixel(mid + 15, yBottomBoundary + 10);
  put_pixel(mid + 15, yBottomBoundary + 7);
  glEnd();
```

```
  verticalWindows(xLeftBoundary + 10 + 10, yBottomBoundary + 25);
  verticalWindows(xRightBoundary - 10 - 10, yBottomBoundary + 25);
  drawDoor(mid);
  drawCircle(mid + 6);

  horizontalWindows(xLeftUpper + 15, upperBottomY + 10);
  horizontalWindows(xRightUpper - 15, upperBottomY + 10);

  double midUpperWindow = ((xRightUpper - 30) + (xLeftUpper + 30)) / 2;
  horizontalWindows(midUpperWindow, upperBottomY + 15);
  drawSemiCircle(midUpperWindow, upperBottomY + 25);
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  glPointSize(3.0);

  drawOutlineFoundation();
  drawInnerDesigns();
  glFlush();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluOrtho2D(-100, 100, -100, 100);

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void init(void) { glClearColor(0, 0, 0, 0); }

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

  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB);

  glutCreateWindow("House Structure");
  init();
```

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



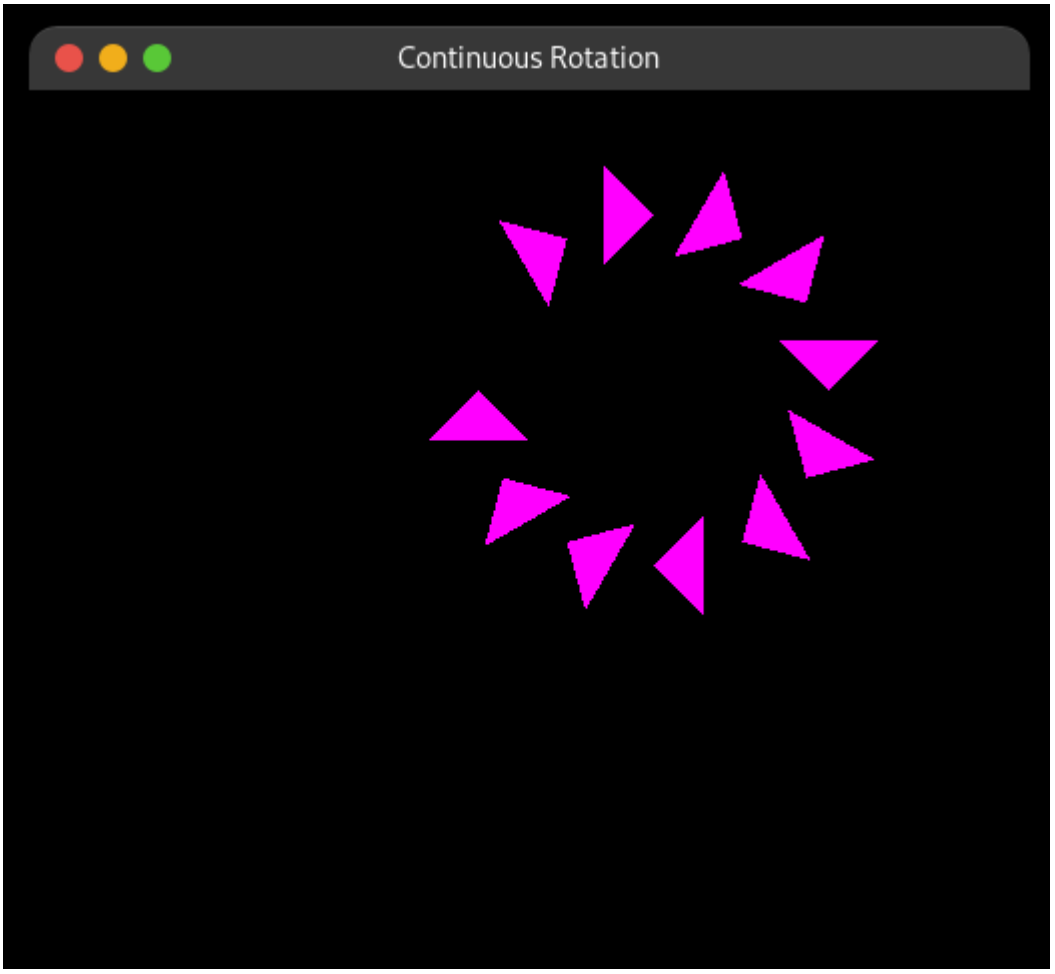
5. Write a program to continuously rotate an object about a pivot point. Hint: Small angles are to be used for each successive rotation, and approximations to the sine and cosine functions are to be used to speed up the calculations. The rotation angle for each step is to be chosen so that the object makes one complete revolution in Ien than 30 seconds. To avoid accumulation of coordinate errors, reset the original coordinate values for the object at the start of each new revolution.

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

```cpp
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>
#include <unordered_map>
#include <vector>

using namespace std;

#define PI 3.14159265
double degree = 0;

vector<vector<double>> cacheMatrix{{{-20, 30, 1}, {0, 30, 1}, {-10, 40,
1}}};

void put_pixel(float r, float g, float b, double x, double y) {
  glColor3f(r, g, b);
  glVertex2d(x, y);
}

vector<vector<double>> multiplyMatrix(vector<vector<double>> a,
                                      vector<vector<double>> b) {
  vector<vector<double>> resMatrix(3, vector<double>(3, 0));
  for (int i = 0; i < 3; ++i) {
    for (int j = 0; j < 3; ++j) {
      for (int k = 0; k < 3; ++k) {
        resMatrix[i][j] += a[i][k] * b[k][j];
      }
    }
  }
  return resMatrix;
}

vector<vector<double>> set_translation_matrix(int dx, int dy) {
  vector<vector<double>> tMatrix{
      {1, 0, 0}, {0, 1, 0}, {(double)dx, (double)dy, 1}};
  return tMatrix;
}

vector<vector<double>> set_rotate_matrix(double degree) {
  vector<vector<double>> rMatrix{
      {cos(degree * PI / 180), sin(degree * PI / 180), 0},
      {sin(-1 * degree * PI / 180), cos(degree * PI / 180), 0},
      {0, 0, 1}};
  return rMatrix;
}
```

```
vector<vector<double>> set_rotate_point_matrix(double degree, int px,
int py) {
  // T(-px, -py).R(degree).T(px, py)
  vector<vector<double>> rPointMatrix(3, vector<double>(3, 0));
  rPointMatrix = multiplyMatrix(set_translation_matrix(-1 * px, -1 *
py),
                                set_rotate_matrix(degree));
  rPointMatrix = multiplyMatrix(rPointMatrix, set_translation_matrix(px,
py));
  return rPointMatrix;
}

void timer(int id) {
  degree += 30;
  if (degree == 360) {
    cacheMatrix = {};
    degree = 0;
  }
  glutPostRedisplay();
}

void placePoints() {
  vector<vector<double>> res = set_rotate_point_matrix(degree, 25, 40);
  vector<vector<double>> temp(3, vector<double>(3, 0));
  vector<vector<double>> points{{{-20, 30, 1}, {0, 30, 1}, {-10, 40,
1}}};
  points = multiplyMatrix(points, res);

  for (int i = 0; i < 3; ++i)
    cacheMatrix.push_back({points[i][0], points[i][1]});
  temp = points;
  for (int i = 0; i < cacheMatrix.size(); ++i)
    put_pixel(1, 0, 1, cacheMatrix[i][0], cacheMatrix[i][1]);
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glLoadIdentity();
  glPointSize(3.0);

  glBegin(GL_TRIANGLES);
  placePoints();
  glEnd();

  glutTimerFunc(1000, timer, 1);
```

```
  glFlush();
}

void reshape(int w, int h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();

  gluOrtho2D(-100, 100, -100, 100);

  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

void init(void) { glClearColor(0, 0, 0, 0); }

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

  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB);

  glutCreateWindow("Continuous Rotation");
  init();
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
}
```


Continuous Rotation