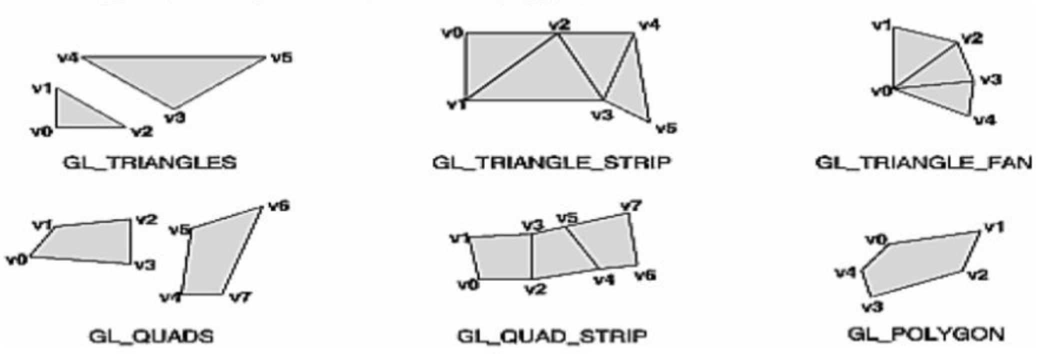
CG Lab 5

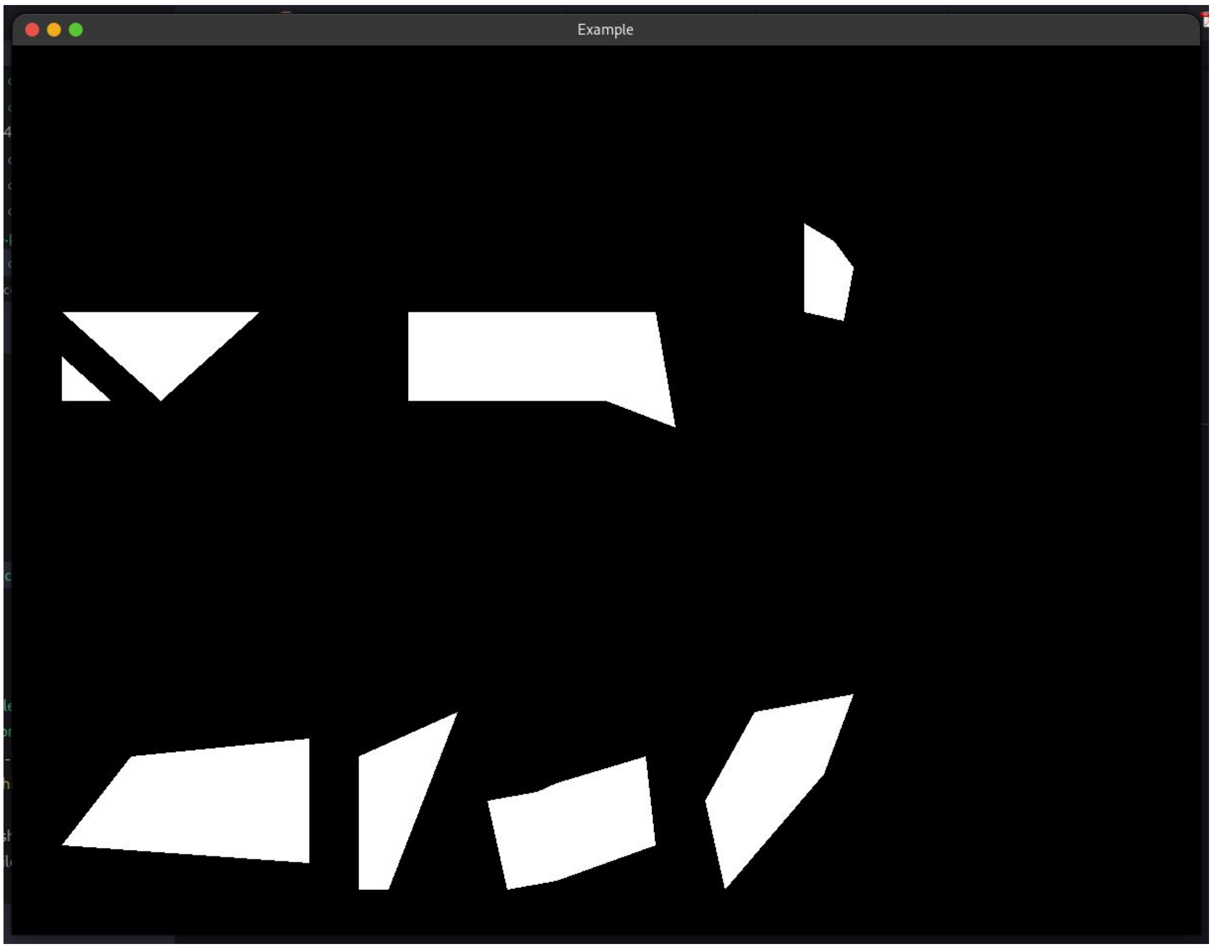
SAHIL BONDRE: U18CO021

1 Write a program to draw the following shapes:
1. Triangles (triples of vertices interpreted as triangles)
2. Triangle Strip (linked strip of triangles)
3. Triangle Fan (linked fan of triangles)
4. Quads (quadruples of vertices interpreted as four sided polygons)
5. Quad Strip (linked strip of quadrilaterals)
6. Polygon (boundary of a simple, convex polygon)



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

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

void draw_triangles() {
  glBegin(GL_TRIANGLES);

  glVertex2d(5, 60);
  glVertex2d(5, 65);
  glVertex2d(10, 60);

  glVertex2d(5, 70);
  glVertex2d(25, 70);
  glVertex2d(15, 60);

  glEnd();
}

void draw_triangle_strip() {
  glBegin(GL_TRIANGLE_STRIP);

  glVertex2d(40, 70);
  glVertex2d(40, 60);
  glVertex2d(50, 70);
  glVertex2d(60, 60);
  glVertex2d(65, 70);
  glVertex2d(67, 57);

  glEnd();
}

void draw_triangle_fan() {
  glBegin(GL_TRIANGLE_FAN);

  glVertex2d(80, 70);
  glVertex2d(80, 80);
  glVertex2d(83, 78);
  glVertex2d(85, 75);
  glVertex2d(84, 69);

  glEnd();
}

void draw_quad() {
  glBegin(GL_QUADS);

  glVertex2d(5, 10);
  glVertex2d(12, 20);
  glVertex2d(30, 22);
  glVertex2d(30, 8);

  glVertex2d(35, 20);
  glVertex2d(45, 25);
  glVertex2d(38, 5);
  glVertex2d(35, 5);

  glEnd();
}

void draw_quad_strip() {
  glBegin(GL_QUAD_STRIP);

  glVertex2d(50, 5);
  glVertex2d(48, 15);
  glVertex2d(55, 6);
  glVertex2d(53, 16);
  glVertex2d(60, 8);
  glVertex2d(55, 17);
  glVertex2d(65, 10);
  glVertex2d(64, 20);

  glEnd();
}

void draw_polygon() {
  glBegin(GL_POLYGON);

  glVertex2d(72, 5);
  glVertex2d(70, 15);
  glVertex2d(75, 25);
  glVertex2d(85, 27);
  glVertex2d(82, 18);

  glEnd();
}

void display() {

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

  draw_triangles();
  draw_triangle_strip();
  draw_triangle_fan();

  draw_quad();
  draw_quad_strip();

  draw_polygon();

  glFlush();
}

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

  gluOrtho2D(0, 120, 0, 100);

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

```
int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitWindowPosition(200, 100);
  glutInitWindowSize(1200, 900);
  glutInitDisplayMode(GLUT_RGB);

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

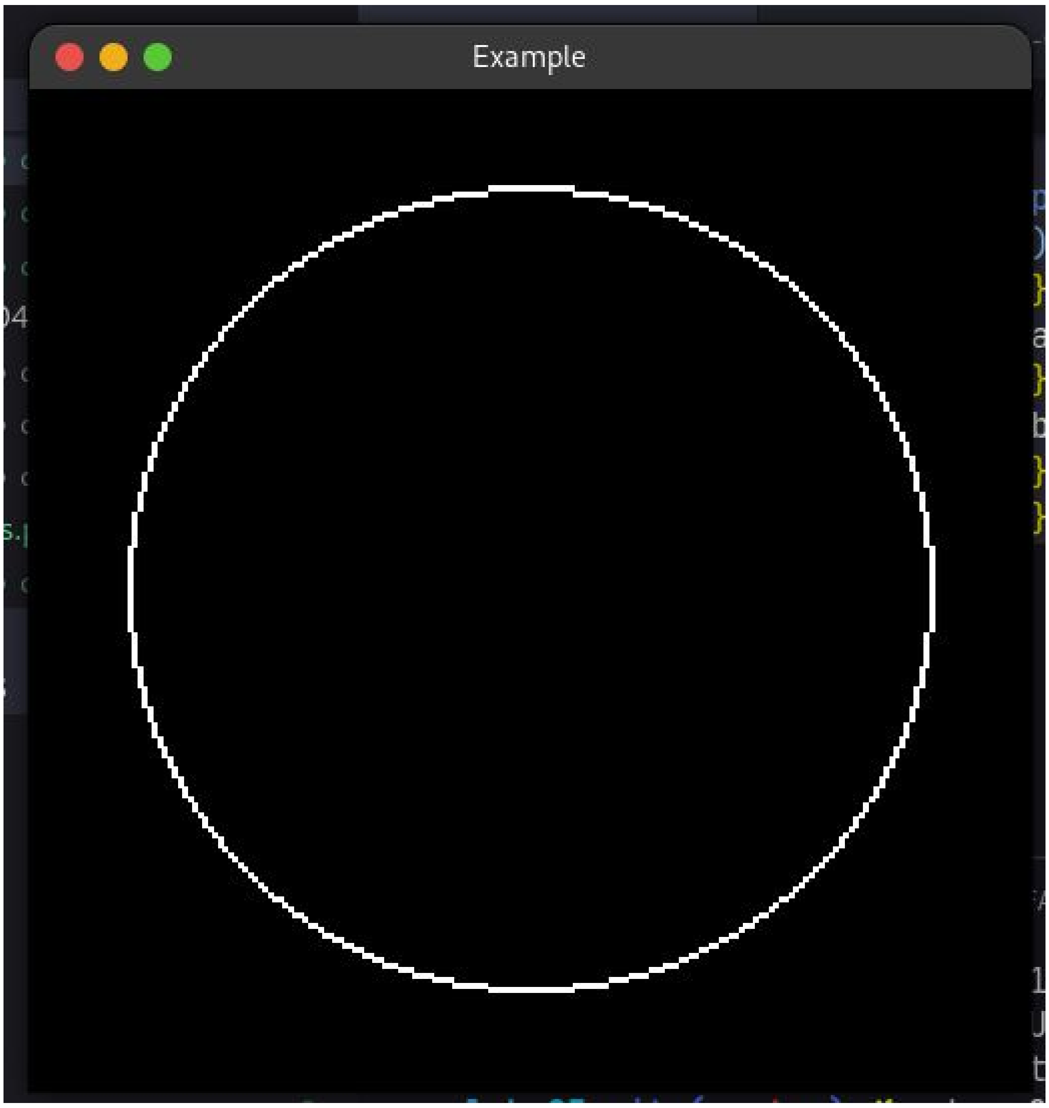


2. Write a menu driven program for following algorithms:

a) **Mid Point Circle generating algorithm**

b) **Mid Point Ellipse generating algorithm**

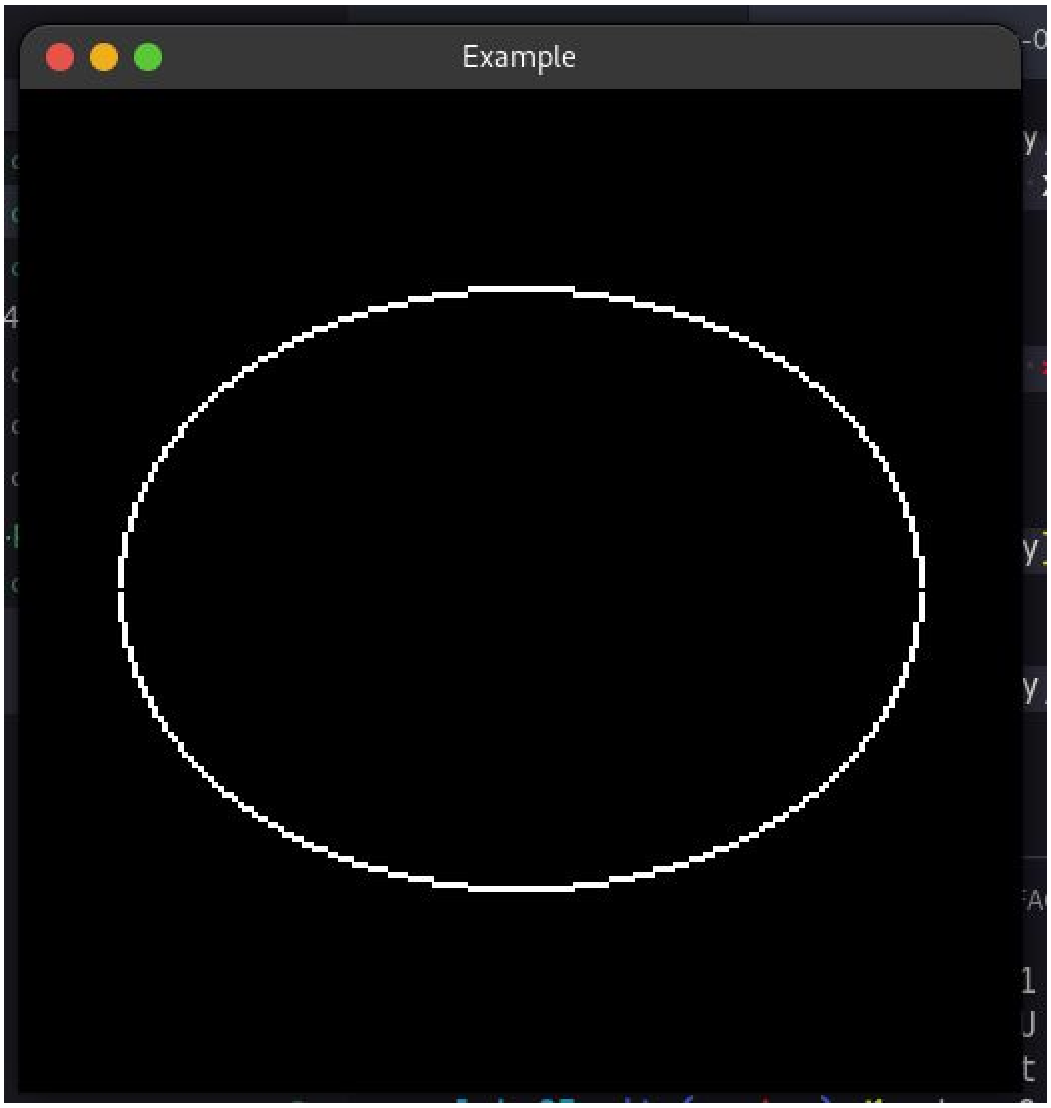
```cpp
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>

using namespace std;

int r = 0;

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

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

void draw_circle() {
  vector<pair<int, int>> coords, temp;
  int x = 0, y = r, p = 5 / 4 - r;

  while (x <= y) {
    coords.push_back({x, y});
    if (p < 0) {
      p += 2 * x + 3;
    } else {
      p += 2 * (x - y) + 5;
      --y;
    }
    ++x;
  }

  // complete octant
  for (auto p : coords) {
    int a = p.first, b = p.second;
    temp.push_back({b, a});
    temp.push_back({-b, a});
    temp.push_back({-b, -a});
    temp.push_back({b, -a});
    temp.push_back({-a, -b});
    temp.push_back({-a, b});
    temp.push_back({a, -b});
  }

  for (auto p : temp) {
    coords.push_back(p);
  }

  for (auto p : temp) {
    coords.push_back(p);
  }

  for (auto p : coords) {
    put_pixel(p.first, p.second);
  }
}

void display() {

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

  glBegin(GL_POINTS);

  draw_circle();

  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 radius of circle: ";
  cin >> r;

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

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



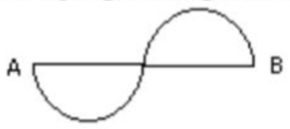
```
#include <GL/glut.h>
#include <bits/stdc++.h>
#include <iostream>
#include <math.h>

using namespace std;

int rx = 0, ry = 0;

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

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

void draw_circle() {
```

```cpp
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});
  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});
  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, b);
  put_pixel(-a, b);
  put_pixel(a, -b);
  put_pixel(-a, -b);
}
```

```
}

void display() {

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

  glBegin(GL_POINTS);

  draw_circle();

  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;

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

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



3 Write a program to generate the following figure :-

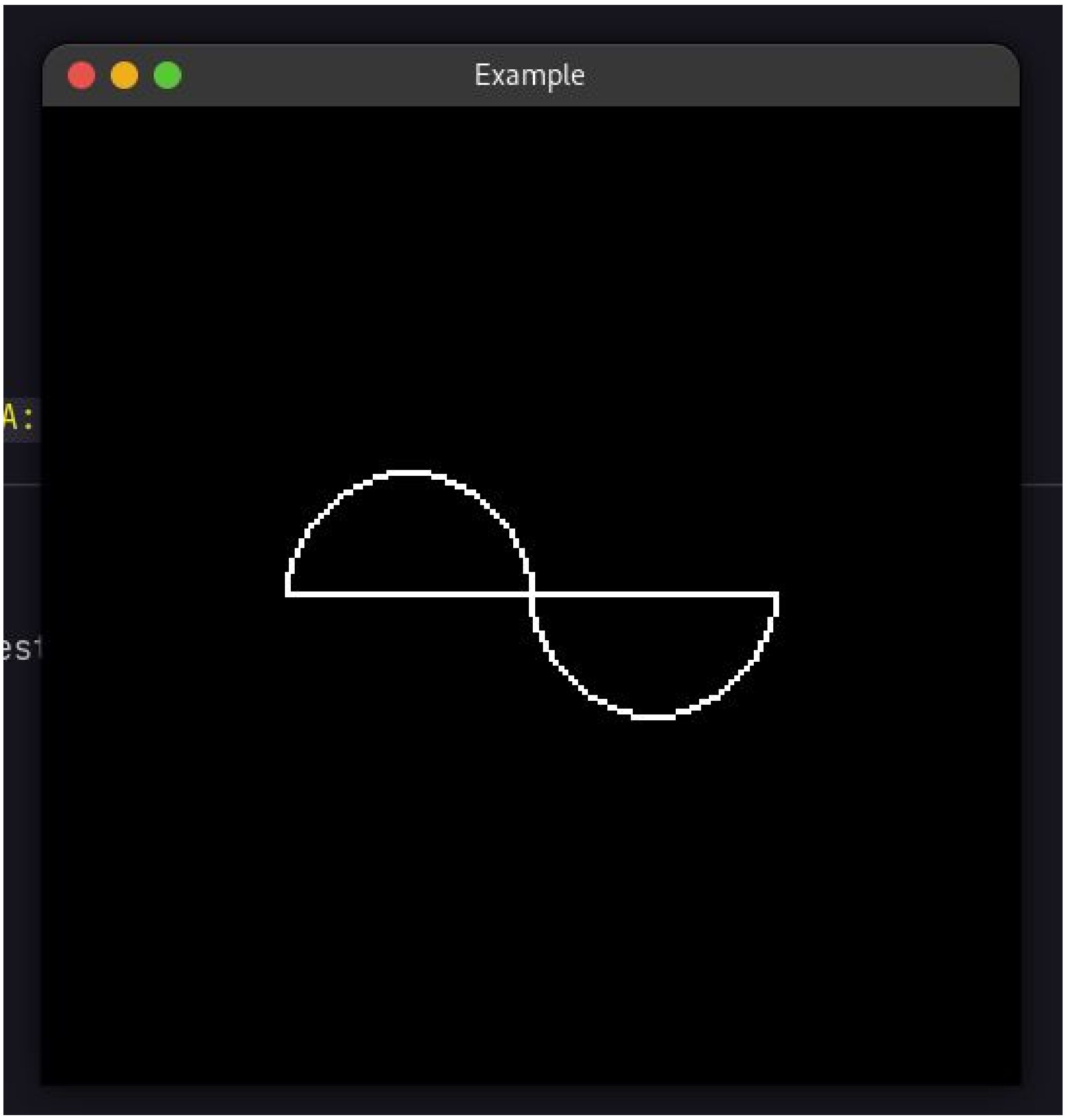


Point A and B is input.

(Use the concept of Mid Point Circle generating algorithm and DDA Line Drawing Algorithm)

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

using namespace std;

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

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

void DDA(int x0, int y0, int x1, int y1) {
  float dx, dy, steps, x, y;
  dx = (float)(x1 - x0);
  dy = (float)(y1 - y0);
  if (abs(dx) >= abs(dy)) {
    steps = abs(dx);
  } else {
    steps = abs(dy);
  }
  dx = dx / steps;
  dy = dy / steps;
  x = (int)x0;
  y = (int)y0;
  int i = 1;
  while (i <= steps) {
    put_pixel(x, y);
    x += dx;
    y += dy;
    i = i + 1;
  }
}

void midpoint1(int xc, int yc, int r) {
  int p, x, y;
  p = 1 - r;
  x = 0;
  y = r;
  while (x <= y) {
    put_pixel(-x + xc, -y + yc);
    put_pixel(-y + xc, -x + yc);
    put_pixel(y + xc, -x + yc);
    put_pixel(x + xc, -y + yc);
    if (p < 0) {
      x++;
      p = p + 2 * x + 3;
    } else {
      y--;
      x++;
      p = p + 2 * x - 2 * y + 5;
    }
  }
}

void midpoint2(int xc, int yc, int r) {
  int p, x, y;
  p = 1 - r;
  x = 0;
  y = r;
  while (x <= y) {
    put_pixel(x + xc, y + yc);
    put_pixel(y + xc, x + yc);
    put_pixel(-x + xc, y + yc);
    put_pixel(-y + xc, x + yc);
    if (p < 0) {
      x++;
      p = p + 2 * x + 3;
    } else {
      y--;
      x++;
      p = p + 2 * x - 2 * y + 5;
    }
  }
}

void display() {

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

  glBegin(GL_POINTS);

  int x0 = 0;
  int x1 = 0;
  int y0 = 0;
  int y1 = 0;
```

```cpp
  cout << "Enter the coordinates point A: ";
  cin >> x0 >> y0;
  cout << "Enter the coordinates point B: ";
  cin >> x1 >> y1;

  DDA(x0, y0, x1, y1);
  int mx = (x0 + x1) / 2;
  int my = (y0 + y1) / 2;
  int c1x = (x0 + mx) / 2;
  int c1y = (y0 + my) / 2;
  int c2x = (x1 + mx) / 2;
  int c2y = (y1 + my) / 2;
  int r1 = sqrt(((mx - c1x) * (mx - c1x)) + ((my - c1y) * (my - c1y)));
  int r2 = sqrt(((mx - c2x) * (mx - c2x)) + ((my - c2y) * (my - c2y)));
  midpoint2(c1x, c1y, r1);
  midpoint1(c2x, c2y, r2);

  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) {

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

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


Example
A:
es