COMPUTER GRAPHICS ASSIGNMENT 02

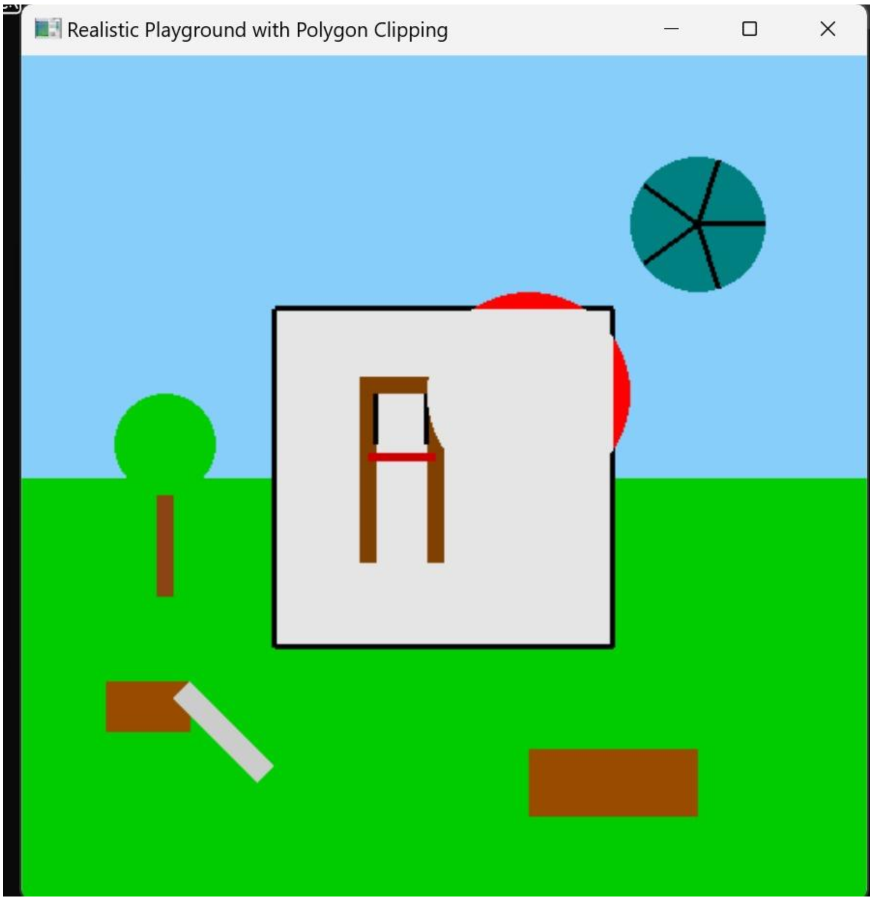
Name : Subi Pinsha.P
Roll No: 2022115020

1. Create a playground scene with five objects (min). Place one object partially inside window and apply polygon clipping algorithm to display only the part of the object lying outside the window.

```
 #include <GL/glut.h>
#include <cmath>
#include <vector>
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

// Define a simple 2D point structure
struct Point {
   float x, y;
};

// Global variables for our play area (clipping rectangle)
float winXmin = 150, winXmax = 350, winYmin = 150, winYmax = 350;

// ---------------------------------------------------------------------
// Sutherland–Hodgman Polygon Clipping Functions (clip against 4 edges)
// ---------------------------------------------------------------------

// Clip against left boundary: x >= xMin
std::vector<Point> clipLeft(const std::vector<Point>& poly, float xMin) {
   std::vector<Point> result;
   int n = poly.size();
   for (int i = 0; i < n; i++) {
      Point curr = poly[i];
      Point prev = poly[(i + n - 1) % n];
      bool currInside = (curr.x >= xMin);
      bool prevInside = (prev.x >= xMin);
      if (prevInside && currInside) {
         result.push_back(curr);
      }
      else if (prevInside && !currInside) {
         float t = (xMin - prev.x) / (curr.x - prev.x);
         Point inter = { xMin, prev.y + t * (curr.y - prev.y) };
         result.push_back(inter);
      }
      else if (!prevInside && currInside) {
         float t = (xMin - prev.x) / (curr.x - prev.x);
         Point inter = { xMin, prev.y + t * (curr.y - prev.y) };
         result.push_back(inter);
         result.push_back(curr);
      }
   }
```

```
    return result;
}

// Clip against right boundary: x <= xMax
std::vector<Point> clipRight(const std::vector<Point>& poly, float xMax) {
    std::vector<Point> result;
    int n = poly.size();
    for (int i = 0; i < n; i++) {
        Point curr = poly[i];
        Point prev = poly[(i + n - 1) % n];
        bool currInside = (curr.x <= xMax);
        bool prevInside = (prev.x <= xMax);
        if (prevInside && currInside) {
            result.push_back(curr);
        }
        else if (prevInside && !currInside) {
            float t = (xMax - prev.x) / (curr.x - prev.x);
            Point inter = { xMax, prev.y + t * (curr.y - prev.y) };
            result.push_back(inter);
        }
        else if (!prevInside && currInside) {
            float t = (xMax - prev.x) / (curr.x - prev.x);
            Point inter = { xMax, prev.y + t * (curr.y - prev.y) };
            result.push_back(inter);
            result.push_back(curr);
        }
    }
    return result;
}

// Clip against bottom boundary: y >= yMin
std::vector<Point> clipBottom(const std::vector<Point>& poly, float yMin) {
    std::vector<Point> result;
    int n = poly.size();
    for (int i = 0; i < n; i++) {
        Point curr = poly[i];
        Point prev = poly[(i + n - 1) % n];
        bool currInside = (curr.y >= yMin);
        bool prevInside = (prev.y >= yMin);
        if (prevInside && currInside) {
            result.push_back(curr);
        }
        else if (prevInside && !currInside) {
            float t = (yMin - prev.y) / (curr.y - prev.y);
            Point inter = { prev.x + t * (curr.x - prev.x), yMin };
            result.push_back(inter);
        }
        else if (!prevInside && currInside) {
            float t = (yMin - prev.y) / (curr.y - prev.y);
            Point inter = { prev.x + t * (curr.x - prev.x), yMin };
            result.push_back(inter);
            result.push_back(curr);
        }
    }
    return result;
}

// Clip against top boundary: y <= yMax
std::vector<Point> clipTop(const std::vector<Point>& poly, float yMax) {
    std::vector<Point> result;
    int n = poly.size();
    for (int i = 0; i < n; i++) {
        Point curr = poly[i];
        Point prev = poly[(i + n - 1) % n];
        bool currInside = (curr.y <= yMax);
        bool prevInside = (prev.y <= yMax);
        if (prevInside && currInside) {
            result.push_back(curr);
        }
        else if (prevInside && !currInside) {
            float t = (yMax - prev.y) / (curr.y - prev.y);
            Point inter = { prev.x + t * (curr.x - prev.x), yMax };
            result.push_back(inter);
        }
        else if (!prevInside && currInside) {
            float t = (yMax - prev.y) / (curr.y - prev.y);
            Point inter = { prev.x + t * (curr.x - prev.x), yMax };
            result.push_back(inter);
            result.push_back(curr);
        }
    }
    return result;
}

// Sutherland–Hodgman clipping: clips polygon 'poly' against a rectangular region.
std::vector<Point> sutherlandHodgmanClip(const std::vector<Point>& poly,
    float xMin, float xMax, float yMin, float yMax) {
    std::vector<Point> output = clipLeft(poly, xMin);
    output = clipRight(output, xMax);
    output = clipBottom(output, yMin);
    output = clipTop(output, yMax);
    return output;
}

// -------------------------
// Drawing Functions for Scene
// -------------------------

// Draw background: sky (upper) and grass (lower)
void drawBackground() {
    // Sky
    glColor3f(0.53f, 0.81f, 0.98f); // light blue
    glBegin(GL_POLYGON);
    glVertex2f(0, 250);
    glVertex2f(500, 250);
    glVertex2f(500, 500);
    glVertex2f(0, 500);
    glEnd();
    // Grass
    glColor3f(0.0f, 0.8f, 0.0f); // green
    glBegin(GL_POLYGON);
    glVertex2f(0, 0);
    glVertex2f(500, 0);
    glVertex2f(500, 250);
    glVertex2f(0, 250);
    glEnd();
}

// Draw the play area as a light-gray rectangle (this is our clipping window)
void drawPlayArea() {
    glColor3f(0.9f, 0.9f, 0.9f); // light gray fill
    glBegin(GL_POLYGON);
    glVertex2f(winXmin, winYmin);
    glVertex2f(winXmax, winYmin);
    glVertex2f(winXmax, winYmax);
    glVertex2f(winXmin, winYmax);
    glEnd();
    glColor3f(0.0f, 0.0f, 0.0f); // black border
    glLineWidth(3.0f);
    glBegin(GL_LINE_LOOP);
    glVertex2f(winXmin, winYmin);
    glVertex2f(winXmax, winYmin);
    glVertex2f(winXmax, winYmax);
    glVertex2f(winXmin, winYmax);
    glEnd();
}

// Draw a slide at the lower left
void drawSlide() {
    // Platform
    glColor3f(0.6f, 0.3f, 0.0f); // brown
    glBegin(GL_POLYGON);
    glVertex2f(50, 100);
    glVertex2f(100, 100);
    glVertex2f(100, 130);
    glVertex2f(50, 130);
    glEnd();
    // Slanted slide surface
    glColor3f(0.8f, 0.8f, 0.8f); // light gray
    glBegin(GL_POLYGON);
    glVertex2f(100, 130);
    glVertex2f(150, 80);
    glVertex2f(140, 70);
    glVertex2f(90, 120);
    glEnd();
}

// Draw a swing set with two posts, a top bar, ropes, and a seat
void drawSwing() {
    // Left post
    glColor3f(0.5f, 0.25f, 0.0f);
    glBegin(GL_POLYGON);
    glVertex2f(200, 200);
    glVertex2f(210, 200);
    glVertex2f(210, 300);
    glVertex2f(200, 300);
    glEnd();
    // Right post
    glBegin(GL_POLYGON);
    glVertex2f(240, 200);
    glVertex2f(250, 200);
    glVertex2f(250, 300);
    glVertex2f(240, 300);
    glEnd();
    // Top bar
    glBegin(GL_POLYGON);
    glVertex2f(200, 300);
    glVertex2f(250, 300);
    glVertex2f(250, 310);
    glVertex2f(200, 310);
    glEnd();
    // Ropes
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
    glVertex2f(210, 300);
    glVertex2f(210, 270);
    glVertex2f(240, 300);
    glVertex2f(240, 270);
    glEnd();
    // Seat (red)
    glColor3f(0.8f, 0.0f, 0.0f);
    glBegin(GL_POLYGON);
    glVertex2f(205, 260);
    glVertex2f(245, 260);
    glVertex2f(245, 265);
    glVertex2f(205, 265);
    glEnd();
}

// Draw a merry-go-round: a circle with spokes
void drawMerryGoRound() {
    glColor3f(0.0f, 0.5f, 0.5f);
    const int segments = 40;
    float cx = 400, cy = 400, r = 40;
    glBegin(GL_POLYGON);
    for (int i = 0; i < segments; i++) {
        float theta = 2.0f * M_PI * i / segments;
        glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
    }
    glEnd();
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
    for (int i = 0; i < segments; i += 8) {
        float theta = 2.0f * M_PI * i / segments;
        glVertex2f(cx, cy);
        glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
    }
    glEnd();
}

// Draw a tree: trunk and circular foliage
void drawTree() {
    // Trunk
    glColor3f(0.55f, 0.27f, 0.07f);
    glBegin(GL_POLYGON);
    glVertex2f(80, 180);
    glVertex2f(90, 180);
    glVertex2f(90, 250);
    glVertex2f(80, 250);
    glEnd();
    // Foliage
    glColor3f(0.0f, 0.8f, 0.0f);
    const int segments = 30;
    float cx = 85, cy = 270, r = 30;
    glBegin(GL_POLYGON);
    for (int i = 0; i < segments; i++) {
        float theta = 2.0f * M_PI * i / segments;
        glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
    }
    glEnd();
}

// Draw a bench
void drawBench() {
    // Seat
    glColor3f(0.6f, 0.3f, 0.0f);
    glBegin(GL_POLYGON);
    glVertex2f(300, 50);
    glVertex2f(400, 50);
    glVertex2f(400, 70);
    glVertex2f(300, 70);
    glEnd();
    // Backrest
    glBegin(GL_POLYGON);
    glVertex2f(300, 70);
    glVertex2f(400, 70);
    glVertex2f(400, 90);
    glVertex2f(300, 90);
    glEnd();
}

// Draw a soccer ball (approximated as a circle) and clip the part inside the play area
void drawBall() {
    std::vector<Point> ballPoly;
    const int segments = 50;
    float cx = 300, cy = 300, r = 60;
    // Create polygon for ball
    for (int i = 0; i < segments; i++) {
        float theta = 2.0f * M_PI * i / segments;
        float x = cx + r * cos(theta);
        float y = cy + r * sin(theta);
        ballPoly.push_back({ x, y });
    }
    // Draw full ball (red)
    glColor3f(1.0f, 0.0f, 0.0f);
    glBegin(GL_POLYGON);
    for (auto p : ballPoly)
        glVertex2f(p.x, p.y);
    glEnd();
    // Clip the ball polygon against the play area
    std::vector<Point> insidePoly = sutherlandHodgmanClip(ballPoly, winXmin, winXmax, winYmin,
winYmax);
    // "Erase" the inside portion by filling it with the play area color (light gray)
    glColor3f(0.9f, 0.9f, 0.9f);
    if (!insidePoly.empty()) {
        glBegin(GL_POLYGON);
        for (auto p : insidePoly)
            glVertex2f(p.x, p.y);
        glEnd();
    }
}

// --------------------------
// Display Callback
// --------------------------
void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Draw background (sky and grass)
    drawBackground();
    // Draw the designated play area (clipping window)
    drawPlayArea();

    // Draw playground objects
    drawSlide();
    drawSwing();
    drawMerryGoRound();
    drawTree();
    drawBench();
    // Draw the soccer ball with clipping applied so only the portion outside the play area shows
    drawBall();

    glFlush();
}

// -------------------------
// Main Function
// -------------------------
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    // Single buffering and RGB mode
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Realistic Playground with Polygon Clipping");

    // Set background clear color (won't be seen because we draw our background)
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
    // Set up a 2D orthographic projection
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 500, 0, 500);

    glutDisplayFunc(display);
    glutMainLoop();
    return 0;
}
```

OUTPUT:



2. Create a classroom scenario with minimum ten objects. Crete an object near the board partially lying inside and apply any line clipping algorithm to clip the portions of the object lying outside the board and display only the part lying inside the board.

```
#include <GL/glut.h>
#include <cmath>
#include <iostream>
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

// -----------------------------
// Global definitions & Globals
// -----------------------------
// Board (clipping region) boundaries
const float boardXmin = 100, boardXmax = 400;
const float boardYmin = 300, boardYmax = 450;

// Cohen-Sutherland region codes
const int INSIDE = 0; // 0000
const int LEFT = 1; // 0001
const int RIGHT = 2; // 0010
const int BOTTOM = 4; // 0100
const int TOP = 8; // 1000

// Compute the region code for a point (x, y)
int computeOutCode(float x, float y, float xmin, float xmax, float ymin, float ymax) {
   int code = INSIDE;
   if (x < xmin) code |= LEFT;
   else if (x > xmax) code |= RIGHT;
   if (y < ymin) code |= BOTTOM;
   else if (y > ymax) code |= TOP;
   return code;
}

// Cohen–Sutherland line clipping algorithm.
// Modifies (x0,y0)-(x1,y1) if the line is partially inside the clip rectangle.
bool cohenSutherlandClip(float& x0, float& y0, float& x1, float& y1,
   float xmin, float xmax, float ymin, float ymax) {
   int code0 = computeOutCode(x0, y0, xmin, xmax, ymin, ymax);
   int code1 = computeOutCode(x1, y1, xmin, xmax, ymin, ymax);
   bool accept = false;

   while (true) {
      if ((code0 | code1) == 0) {
         // Both endpoints inside: accept the line.
         accept = true;
         break;
      }
      else if (code0 & code1) {
         // Both endpoints share an outside zone (trivial reject).
         break;
      }
      else {
         // At least one endpoint is outside the clip rectangle.
         int outCode = code0 ? code0 : code1;
         float x, y;
         if (outCode & TOP) {
            // Point is above the clip rectangle.
            x = x0 + (x1 - x0) * (boardYmax - y0) / (y1 - y0);
            y = boardYmax;
         }
         else if (outCode & BOTTOM) {
            // Point is below the clip rectangle.
            x = x0 + (x1 - x0) * (boardYmin - y0) / (y1 - y0);
            y = boardYmin;
         }
         else if (outCode & RIGHT) {
            // Point is to the right of clip rectangle.
            y = y0 + (y1 - y0) * (boardXmax - x0) / (x1 - x0);
            x = boardXmax;
         }
         else if (outCode & LEFT) {
            // Point is to the left of clip rectangle.
            y = y0 + (y1 - y0) * (boardXmin - x0) / (x1 - x0);
            x = boardXmin;
         }
         // Now update the point that was outside the rectangle.
         if (outCode == code0) {
            x0 = x; y0 = y;
            code0 = computeOutCode(x0, y0, xmin, xmax, ymin, ymax);
         }
         else {
            x1 = x; y1 = y;
            code1 = computeOutCode(x1, y1, xmin, xmax, ymin, ymax);
         }
      }
   }
   return accept;
}

// ---------------------
// Object Drawing Functions
// ---------------------

// Draw the black board (which serves as the clipping region for the clock)
void drawBlackBoard() {
   glColor3f(0.0f, 0.5f, 0.0f);  // dark green board
   glBegin(GL_POLYGON);
   glVertex2f(boardXmin, boardYmin);
   glVertex2f(boardXmax, boardYmin);
   glVertex2f(boardXmax, boardYmax);
   glVertex2f(boardXmin, boardYmax);
   glEnd();

   // Draw board frame
   glColor3f(0.0f, 0.0f, 0.0f);
   glLineWidth(2.0f);
   glBegin(GL_LINE_LOOP);
   glVertex2f(boardXmin, boardYmin);
   glVertex2f(boardXmax, boardYmin);
   glVertex2f(boardXmax, boardYmax);
   glVertex2f(boardXmin, boardYmax);
   glEnd();
}

// Draw the clock with its circular boundary (approximated by line segments)
// and clip each segment so that only portions inside the board are drawn.
void drawClippedClock() {
   // Clock parameters: placed so that it is partially outside the board.
   float cx = 80, cy = 400; // center of clock
   float r = 50;          // radius of clock
   const int segments = 50;
   glColor3f(1.0f, 0.0f, 0.0f);  // red clock outline
```

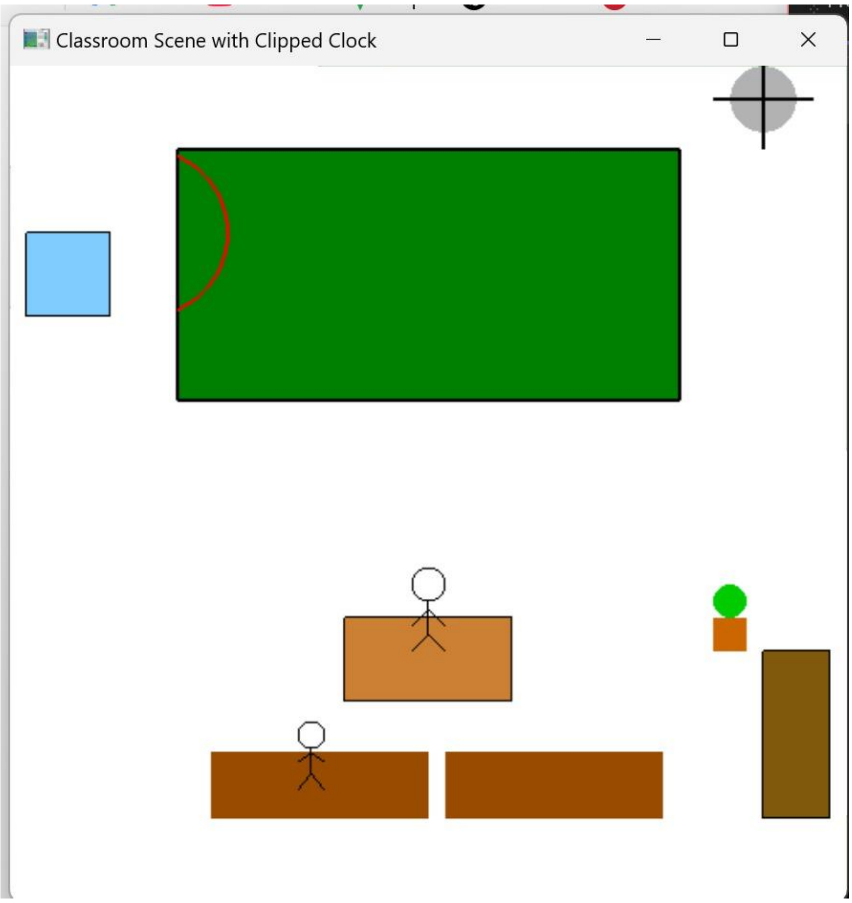
```
    glBegin(GL_LINES);
    for (int i = 0; i < segments; i++) {
        float theta1 = 2.0f * M_PI * i / segments;
        float theta2 = 2.0f * M_PI * (i + 1) / segments;
        float x0 = cx + r * cos(theta1);
        float y0 = cy + r * sin(theta1);
        float x1 = cx + r * cos(theta2);
        float y1 = cy + r * sin(theta2);

        // Copy endpoints for clipping
        float clipX0 = x0, clipY0 = y0;
        float clipX1 = x1, clipY1 = y1;

        if (cohenSutherlandClip(clipX0, clipY0, clipX1, clipY1,
            boardXmin, boardXmax, boardYmin, boardYmax))
        {
            glVertex2f(clipX0, clipY0);
            glVertex2f(clipX1, clipY1);
        }
    }
    glEnd();
}

// Draw a fan on the ceiling.
void drawFan() {
    // Fan body (circle) at (450,480) with radius 20.
    float cx = 450, cy = 480, r = 20;
    glColor3f(0.7f, 0.7f, 0.7f);
    glBegin(GL_POLYGON);
    int segments = 20;
    for (int i = 0; i < segments; i++) {
        float theta = 2.0f * M_PI * i / segments;
        glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
    }
    glEnd();

    // Fan blades (simple lines)
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
    glVertex2f(cx, cy);
    glVertex2f(cx, cy + r + 10);

    glVertex2f(cx, cy);
    glVertex2f(cx + r + 10, cy);

    glVertex2f(cx, cy);
    glVertex2f(cx, cy - r - 10);

    glVertex2f(cx, cy);
    glVertex2f(cx - r - 10, cy);
```

```
   glEnd();
}

// Draw a window on the left wall.
void drawWindow() {
   glColor3f(0.5f, 0.8f, 1.0f);  // light blue glass
   glBegin(GL_POLYGON);
   glVertex2f(10, 350);
   glVertex2f(60, 350);
   glVertex2f(60, 400);
   glVertex2f(10, 400);
   glEnd();

   // Window frame
   glColor3f(0.0f, 0.0f, 0.0f);
   glLineWidth(1.0f);
   glBegin(GL_LINE_LOOP);
   glVertex2f(10, 350);
   glVertex2f(60, 350);
   glVertex2f(60, 400);
   glVertex2f(10, 400);
   glEnd();
}

// Draw benches for the students.
void drawBenches() {
   glColor3f(0.6f, 0.3f, 0.0f);  // brown bench
   // First bench
   glBegin(GL_POLYGON);
   glVertex2f(120, 50);
   glVertex2f(250, 50);
   glVertex2f(250, 90);
   glVertex2f(120, 90);
   glEnd();
   // Second bench
   glBegin(GL_POLYGON);
   glVertex2f(260, 50);
   glVertex2f(390, 50);
   glVertex2f(390, 90);
   glVertex2f(260, 90);
   glEnd();
}

// Draw the teacher’s table.
void drawTeacherTable() {
   glColor3f(0.8f, 0.5f, 0.2f);  // table color
   glBegin(GL_POLYGON);
   glVertex2f(200, 120);
   glVertex2f(300, 120);
   glVertex2f(300, 170);
   glVertex2f(200, 170);
```

```
    glEnd();

    // Table outline
    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINE_LOOP);
    glVertex2f(200, 120);
    glVertex2f(300, 120);
    glVertex2f(300, 170);
    glVertex2f(200, 170);
    glEnd();
}

// Draw a stickman representing the teacher.
void drawTeacherStickman() {
    // Head: circle centered at (250,190) with radius 10.
    float cx = 250, cy = 190, r = 10;
    glColor3f(0.0f, 0.0f, 0.0f);
    int segments = 20;
    glBegin(GL_LINE_LOOP);
    for (int i = 0; i < segments; i++) {
        float theta = 2.0f * M_PI * i / segments;
        glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
    }
    glEnd();
    // Body
    glBegin(GL_LINES);
    glVertex2f(cx, cy - r);
    glVertex2f(cx, cy - r - 20);
    glEnd();
    // Arms
    glBegin(GL_LINES);
    glVertex2f(cx, cy - r - 5);
    glVertex2f(cx - 10, cy - r - 15);
    glVertex2f(cx, cy - r - 5);
    glVertex2f(cx + 10, cy - r - 15);
    glEnd();
    // Legs
    glBegin(GL_LINES);
    glVertex2f(cx, cy - r - 20);
    glVertex2f(cx - 10, cy - r - 30);
    glVertex2f(cx, cy - r - 20);
    glVertex2f(cx + 10, cy - r - 30);
    glEnd();
}

// Draw a stickman representing a student.
void drawStudentStickman() {
    // Head: circle centered at (180,100) with radius 8.
    float cx = 180, cy = 100, r = 8;
    glColor3f(0.0f, 0.0f, 0.0f);
    int segments = 20;
```

```
  glBegin(GL_LINE_LOOP);
  for (int i = 0; i < segments; i++) {
    float theta = 2.0f * M_PI * i / segments;
    glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
  }
  glEnd();
  // Body
  glBegin(GL_LINES);
  glVertex2f(cx, cy - r);
  glVertex2f(cx, cy - r - 15);
  glEnd();
  // Arms
  glBegin(GL_LINES);
  glVertex2f(cx, cy - r - 3);
  glVertex2f(cx - 8, cy - r - 8);
  glVertex2f(cx, cy - r - 3);
  glVertex2f(cx + 8, cy - r - 8);
  glEnd();
  // Legs
  glBegin(GL_LINES);
  glVertex2f(cx, cy - r - 15);
  glVertex2f(cx - 8, cy - r - 25);
  glVertex2f(cx, cy - r - 15);
  glVertex2f(cx + 8, cy - r - 25);
  glEnd();
}

// Draw a small plant in a pot.
void drawPlant() {
  // Pot
  glColor3f(0.8f, 0.4f, 0.0f);
  glBegin(GL_POLYGON);
  glVertex2f(420, 150);
  glVertex2f(440, 150);
  glVertex2f(440, 170);
  glVertex2f(420, 170);
  glEnd();
  // Plant (a small green circle)
  float cx = 430, cy = 180, r = 10;
  glColor3f(0.0f, 0.8f, 0.0f);
  int segments = 20;
  glBegin(GL_POLYGON);
  for (int i = 0; i < segments; i++) {
    float theta = 2.0f * M_PI * i / segments;
    glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
  }
  glEnd();
}

// Draw a door on the right side of the classroom.
void drawDoor() {
  glColor3f(0.5f, 0.35f, 0.05f);
  glBegin(GL_POLYGON);
  glVertex2f(450, 50);
  glVertex2f(490, 50);
  glVertex2f(490, 150);
  glVertex2f(450, 150);
  glEnd();

  // Door frame
  glColor3f(0.0f, 0.0f, 0.0f);
  glBegin(GL_LINE_LOOP);
  glVertex2f(450, 50);
  glVertex2f(490, 50);
  glVertex2f(490, 150);
  glVertex2f(450, 150);
  glEnd();
}

// ---------------------
// Display and Main Loop
// ---------------------
void display() {
  glClear(GL_COLOR_BUFFER_BIT);

  // Draw classroom objects:
  drawBlackBoard();      // (clipping region for clock)
  drawClippedClock();    // clock (only the part inside the board is drawn)
  drawFan();
  drawWindow();
  drawBenches();
  drawTeacherTable();
  drawTeacherStickman();
  drawStudentStickman();
  drawPlant();
  drawDoor();

  glFlush();
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  // Single buffering and RGB color mode
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("Classroom Scene with Clipped Clock");

  // White background
  glClearColor(1.0f, 1.0f, 1.0f, 1.0f);
  // 2D orthogonal projection
  glMatrixMode(GL_PROJECTION);
  gluOrtho2D(0, 500, 0, 500);

  glutDisplayFunc(display);
  glutMainLoop();
  return 0;
}
```

OUTPUT:



3.Apply text clipping algorithm to portions of text available in the black board created in the previous question.

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

// Define M_PI if not already defined
#ifndef M_PI
#define M_PI 3.14159265358979323846
#endif

// ----------------------------
// Global Definitions
// -----------------------------
const float boardXmin = 100, boardXmax = 400;
const float boardYmin = 300, boardYmax = 450;

// Cohen–Sutherland region codes
const int INSIDE = 0; // 0000
const int LEFT = 1; // 0001
const int RIGHT = 2; // 0010
const int BOTTOM = 4; // 0100
const int TOP = 8; // 1000

// Compute the region code for a point (x, y)
int computeOutCode(float x, float y, float xmin, float xmax, float ymin, float ymax) {
    int code = INSIDE;
    if (x < xmin) code |= LEFT;
    else if (x > xmax) code |= RIGHT;
    if (y < ymin) code |= BOTTOM;
    else if (y > ymax) code |= TOP;
    return code;
}

// Cohen–Sutherland line clipping algorithm.
// Modifies the endpoints (x0,y0)–(x1,y1) if the line is partially inside the clip rectangle.
bool cohenSutherlandClip(float& x0, float& y0, float& x1, float& y1,
    float xmin, float xmax, float ymin, float ymax) {
    int code0 = computeOutCode(x0, y0, xmin, xmax, ymin, ymax);
    int code1 = computeOutCode(x1, y1, xmin, xmax, ymin, ymax);
    bool accept = false;

    while (true) {
        if ((code0 | code1) == 0) { // both points inside
            accept = true;
            break;
        }
        else if (code0 & code1) { // both points share an outside zone
            break;
        }
        else {
            int outCode = code0 ? code0 : code1;
            float x, y;
            if (outCode & TOP) {
                x = x0 + (x1 - x0) * (boardYmax - y0) / (y1 - y0);
                y = boardYmax;
            }
            else if (outCode & BOTTOM) {
                x = x0 + (x1 - x0) * (boardYmin - y0) / (y1 - y0);
                y = boardYmin;
            }
            else if (outCode & RIGHT) {
                y = y0 + (y1 - y0) * (boardXmax - x0) / (x1 - x0);
                x = boardXmax;
            }
            else if (outCode & LEFT) {
                y = y0 + (y1 - y0) * (boardXmin - x0) / (x1 - x0);
                x = boardXmin;
            }
            if (outCode == code0) {
                x0 = x; y0 = y;
                code0 = computeOutCode(x0, y0, xmin, xmax, ymin, ymax);
            }
            else {
                x1 = x; y1 = y;
                code1 = computeOutCode(x1, y1, xmin, xmax, ymin, ymax);
            }
        }
    }
    return accept;
}

// -----------------------------
// Object Drawing Functions
// -----------------------------

// Draw the black board (used as clipping region for clock and text)
void drawBlackBoard() {
    glColor3f(0.0f, 0.5f, 0.0f);  // dark green board
    glBegin(GL_POLYGON);
    glVertex2f(boardXmin, boardYmin);
    glVertex2f(boardXmax, boardYmin);
    glVertex2f(boardXmax, boardYmax);
    glVertex2f(boardXmin, boardYmax);
    glEnd();

    // Draw the board frame
    glColor3f(0.0f, 0.0f, 0.0f);
    glLineWidth(2.0f);
    glBegin(GL_LINE_LOOP);
    glVertex2f(boardXmin, boardYmin);
    glVertex2f(boardXmax, boardYmin);
    glVertex2f(boardXmax, boardYmax);
    glVertex2f(boardXmin, boardYmax);
    glEnd();
}

// Draw the clock with its circular outline, clipping each line segment against the board.
void drawClippedClock() {
    float cx = 80, cy = 400; // clock center (partially outside board)
    float r = 50;
    const int segments = 50;
    glColor3f(1.0f, 0.0f, 0.0f);  // red clock outline

    glBegin(GL_LINES);
    for (int i = 0; i < segments; i++) {
        float theta1 = 2.0f * M_PI * i / segments;
        float theta2 = 2.0f * M_PI * (i + 1) / segments;
        float x0 = cx + r * cos(theta1);
        float y0 = cy + r * sin(theta1);
        float x1 = cx + r * cos(theta2);
        float y1 = cy + r * sin(theta2);

        // Copy endpoints for clipping
        float clipX0 = x0, clipY0 = y0;
        float clipX1 = x1, clipY1 = y1;

        if (cohenSutherlandClip(clipX0, clipY0, clipX1, clipY1,
            boardXmin, boardXmax, boardYmin, boardYmax)) {
            glVertex2f(clipX0, clipY0);
            glVertex2f(clipX1, clipY1);
        }
    }
    glEnd();
}

// Draw a fan on the ceiling.
void drawFan() {
    float cx = 450, cy = 480, r = 20;
    glColor3f(0.7f, 0.7f, 0.7f);
    glBegin(GL_POLYGON);
    const int segments = 20;
    for (int i = 0; i < segments; i++) {
        float theta = 2.0f * M_PI * i / segments;
        glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
    }
    glEnd();

    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINES);
    glVertex2f(cx, cy);
    glVertex2f(cx, cy + r + 10);
    glVertex2f(cx, cy);
    glVertex2f(cx + r + 10, cy);
    glVertex2f(cx, cy);
    glVertex2f(cx, cy - r - 10);
    glVertex2f(cx, cy);
    glVertex2f(cx - r - 10, cy);
    glEnd();
}

// Draw a window on the left wall.
void drawWindow() {
    glColor3f(0.5f, 0.8f, 1.0f);  // light blue glass
    glBegin(GL_POLYGON);
    glVertex2f(10, 350);
    glVertex2f(60, 350);
    glVertex2f(60, 400);
    glVertex2f(10, 400);
    glEnd();

    glColor3f(0.0f, 0.0f, 0.0f);
    glLineWidth(1.0f);
    glBegin(GL_LINE_LOOP);
    glVertex2f(10, 350);
    glVertex2f(60, 350);
    glVertex2f(60, 400);
    glVertex2f(10, 400);
    glEnd();
}

// Draw benches for the students.
void drawBenches() {
    glColor3f(0.6f, 0.3f, 0.0f);
    // First bench
    glBegin(GL_POLYGON);
    glVertex2f(120, 50);
    glVertex2f(250, 50);
    glVertex2f(250, 90);
    glVertex2f(120, 90);
    glEnd();
    // Second bench
    glBegin(GL_POLYGON);
    glVertex2f(260, 50);
    glVertex2f(390, 50);
    glVertex2f(390, 90);
    glVertex2f(260, 90);
    glEnd();
}

// Draw the teacher’s table.
void drawTeacherTable() {
    glColor3f(0.8f, 0.5f, 0.2f);
    glBegin(GL_POLYGON);
    glVertex2f(200, 120);
    glVertex2f(300, 120);
    glVertex2f(300, 170);
    glVertex2f(200, 170);
    glEnd();

    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINE_LOOP);
    glVertex2f(200, 120);
    glVertex2f(300, 120);
    glVertex2f(300, 170);
    glVertex2f(200, 170);
    glEnd();
```

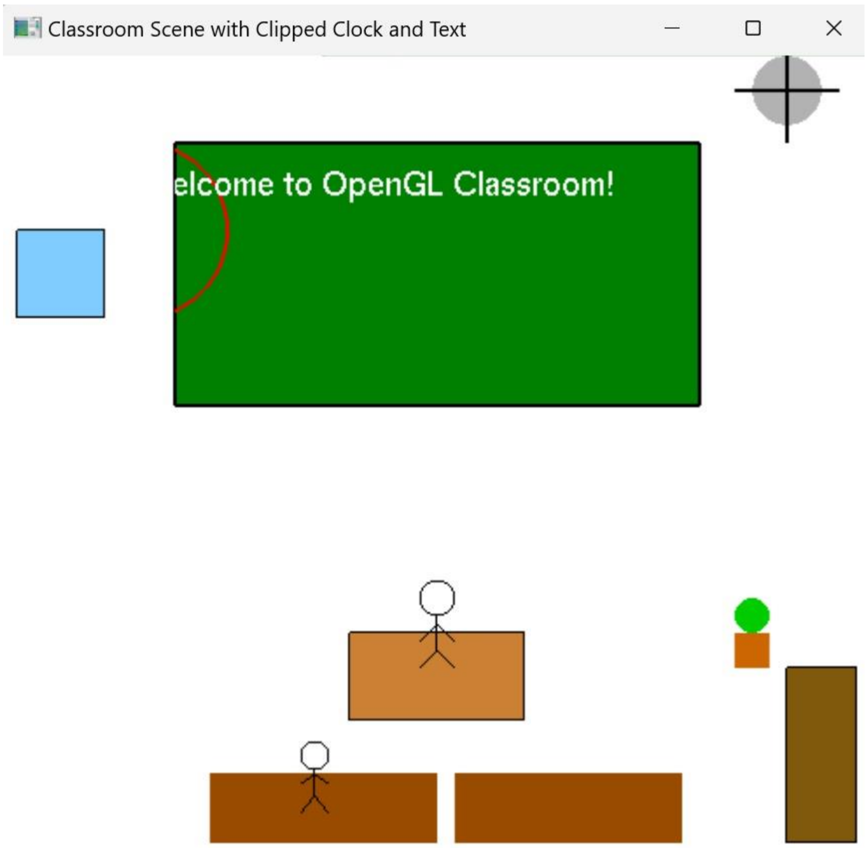
```
}

// Draw a stickman representing the teacher.
void drawTeacherStickman() {
   float cx = 250, cy = 190, r = 10;
   glColor3f(0.0f, 0.0f, 0.0f);
   int segments = 20;
   glBegin(GL_LINE_LOOP);
   for (int i = 0; i < segments; i++) {
      float theta = 2.0f * M_PI * i / segments;
      glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
   }
   glEnd();
   // Body
   glBegin(GL_LINES);
   glVertex2f(cx, cy - r);
   glVertex2f(cx, cy - r - 20);
   glEnd();
   // Arms
   glBegin(GL_LINES);
   glVertex2f(cx, cy - r - 5);
   glVertex2f(cx - 10, cy - r - 15);
   glVertex2f(cx, cy - r - 5);
   glVertex2f(cx + 10, cy - r - 15);
   glEnd();
   // Legs
   glBegin(GL_LINES);
   glVertex2f(cx, cy - r - 20);
   glVertex2f(cx - 10, cy - r - 30);
   glVertex2f(cx, cy - r - 20);
   glVertex2f(cx + 10, cy - r - 30);
   glEnd();
}

// Draw a stickman representing a student.
void drawStudentStickman() {
   float cx = 180, cy = 100, r = 8;
   glColor3f(0.0f, 0.0f, 0.0f);
   int segments = 20;
   glBegin(GL_LINE_LOOP);
   for (int i = 0; i < segments; i++) {
      float theta = 2.0f * M_PI * i / segments;
      glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
   }
   glEnd();
   // Body
   glBegin(GL_LINES);
   glVertex2f(cx, cy - r);
   glVertex2f(cx, cy - r - 15);
   glEnd();
   // Arms
```

```
    glBegin(GL_LINES);
    glVertex2f(cx, cy - r - 3);
    glVertex2f(cx - 8, cy - r - 8);
    glVertex2f(cx, cy - r - 3);
    glVertex2f(cx + 8, cy - r - 8);
    glEnd();
    // Legs
    glBegin(GL_LINES);
    glVertex2f(cx, cy - r - 15);
    glVertex2f(cx - 8, cy - r - 25);
    glVertex2f(cx, cy - r - 15);
    glVertex2f(cx + 8, cy - r - 25);
    glEnd();
}

// Draw a small plant in a pot.
void drawPlant() {
    // Pot
    glColor3f(0.8f, 0.4f, 0.0f);
    glBegin(GL_POLYGON);
    glVertex2f(420, 150);
    glVertex2f(440, 150);
    glVertex2f(440, 170);
    glVertex2f(420, 170);
    glEnd();
    // Plant (circle)
    float cx = 430, cy = 180, r = 10;
    glColor3f(0.0f, 0.8f, 0.0f);
    int segmentsPlant = 20;
    glBegin(GL_POLYGON);
    for (int i = 0; i < segmentsPlant; i++) {
        float theta = 2.0f * M_PI * i / segmentsPlant;
        glVertex2f(cx + r * cos(theta), cy + r * sin(theta));
    }
    glEnd();
}

// Draw a door on the right side.
void drawDoor() {
    glColor3f(0.5f, 0.35f, 0.05f);
    glBegin(GL_POLYGON);
    glVertex2f(450, 50);
    glVertex2f(490, 50);
    glVertex2f(490, 150);
    glVertex2f(450, 150);
    glEnd();

    glColor3f(0.0f, 0.0f, 0.0f);
    glBegin(GL_LINE_LOOP);
    glVertex2f(450, 50);
    glVertex2f(490, 50);
```

```
  glVertex2f(490, 150);
  glVertex2f(450, 150);
  glEnd();
}

// -----------------------------
// Text Clipping Function
// -----------------------------
// Draws a text string using GLUT bitmap fonts, clipping the output to the black board.
void drawClippedText(const char* text, float x, float y) {
  glEnable(GL_SCISSOR_TEST);
  // Set scissor region to match the board (scissor expects integer values)
  glScissor((int)boardXmin, (int)boardYmin, (int)(boardXmax - boardXmin), (int)(boardYmax -
boardYmin));

  glColor3f(1.0f, 1.0f, 1.0f); // white text
  glRasterPos2f(x, y);
  for (int i = 0; i < (int)strlen(text); i++) {
    glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, text[i]);
  }

  glDisable(GL_SCISSOR_TEST);
}

// -----------------------------
// Display Callback
// -----------------------------
void display() {
  glClear(GL_COLOR_BUFFER_BIT);

  // Draw classroom objects
  drawBlackBoard();

  // Draw the clock (line clipping applied)
  drawClippedClock();

  // Draw text on the board.
  // The starting x position is chosen so that part of the text falls outside the board.
  drawClippedText("Welcome to OpenGL Classroom!", 80, 420);

  drawFan();
  drawWindow();
  drawBenches();
  drawTeacherTable();
  drawTeacherStickman();
  drawStudentStickman();
  drawPlant();
  drawDoor();

  glFlush();
}
```

```
// -----------------------------
// Main Function
// -----------------------------
int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("Classroom Scene with Clipped Clock and Text");

   glClearColor(1.0, 1.0, 1.0, 1.0);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D(0, 500, 0, 500);

   glutDisplayFunc(display);
   glutMainLoop();
   return 0;
}
```

OUTPUT:



Classroom Scene with Clipped Clock and Text