```cpp
// Copyright 2023 Thomas O’Connor
#include "PTree.hpp"

PTree::PTree(double L, int N) : _L(L), _N(N) {
    if (N < 1 || L <= 0.0) throw std::out_of_range("Invalid arguments");
}

void PTree::draw(sf::RenderTarget& target, sf::RenderStates states) const {
    if (_N) {
        // first square
        RectangleShape base(Vector2f(_L, _L));
        base.setFillColor(sf::Color::Green);
        base.setOrigin(Vector2f(_L/2, _L));
        base.setPosition(Vector2f(_L*3, _L*4));
        target.draw(base);
        // create a pair of Vector2f that locate the upper points
        pair<Vector2f, Vector2f> newPoints;
        newPoints = pair(Vector2f(base.getPosition().x-_L/2, base.getPosition().y-_L),
            Vector2f(base.getPosition().x+_L/2, base.getPosition().y-_L));
        // call the recursive draw function with decremented depth and 0 angle
        pTree(target, newPoints, _N-1, 0.0f);
    }
}

// Recursive draw function
void PTree::pTree(sf::RenderTarget& target,
    pair<Vector2f, Vector2f> newPoints, int depthN,
    float angleR) const {
    // if more depth exists: draw
    if (depthN) {
        // Find object length and object angles
        // drawIndicators(target, newPoints); // shows the points where new squares will be drawn
        double objLength = _L*pow(sqrt(2) / 2, _N-depthN);
        float leftAngleR = angleR - 45.f, rightAngleR = angleR + 45.f;
        // Draw two rectangles given the newPoints
        RectangleShape leftObj(Vector2f(objLength, objLength));
        RectangleShape rightObj(Vector2f(objLength, objLength));
        switch (depthN % 3) {
            case 0:
            leftObj.setFillColor(sf::Color::Red);
            rightObj.setFillColor(sf::Color::Red);
            break;
            case 1:
            leftObj.setFillColor(sf::Color::Yellow);
            rightObj.setFillColor(sf::Color::Yellow);
            break;
            case 2:
            leftObj.setFillColor(sf::Color::Blue);
            rightObj.setFillColor(sf::Color::Blue);
            break;
        }
        leftObj.setOrigin(0, objLength); rightObj.setOrigin(objLength, objLength);
        leftObj.rotate(leftAngleR); rightObj.rotate(rightAngleR);
        leftObj.setPosition(newPoints.first); rightObj.setPosition(newPoints.second);
        target.draw(leftObj); target.draw(rightObj);

        // Use given information to calculate the location of the next 4 points
        pair<Vector2f, Vector2f> newPointsL, newPointsR;
        // Get the global bounds of the rectangles
        sf::FloatRect boundsL = leftObj.getGlobalBounds(), boundsR = rightObj.getGlobalBounds();
        // Calculate the center point of the rectangles
        Vector2f centerL(boundsL.left + boundsL.width / 2.f, boundsL.top + boundsL.height / 2.f);
        Vector2f centerR(boundsR.left + boundsR.width / 2.f, boundsR.top + boundsR.height / 2.f);
        // drawIndicator(target, centerL); drawIndicator(target, centerR); // shows the centerpoints

        // Calculate the distance between the center point and each corner of the rectangles
        float distance = objLength * sqrt(2.f) / 2.f;
        // Calculate the angle between the center point and each corner of the left rectangle
        float angleLeft = atan2(boundsL.top - centerL.y, boundsL.left - centerL.x);
        float angleRight = atan2(boundsL.top - centerL.y, boundsL.left + boundsL.width - centerL.x);

        // Add the angle of rotation and calculate the coordinates after rotation
        Vector2f upperLeftL(centerL.x + distance * cos(angleLeft + leftAngleR * M_PI / 180.f),
            centerL.y + distance * sin(angleLeft + leftAngleR * M_PI / 180.f));
        Vector2f upperRightL(centerL.x + distance * cos(angleRight + leftAngleR * M_PI / 180.f),
            centerL.y + distance * sin(angleRight + leftAngleR * M_PI / 180.f));
        // input these new coordinates into pair
        newPointsL = pair(upperLeftL, upperRightL);

        // Calculate the angle between the center point and the corners of the other rectangle
        angleLeft = atan2(boundsR.top - centerR.y, boundsR.left - centerR.x);
        angleRight = atan2(boundsR.top - centerR.y, boundsR.left + boundsR.width - centerR.x);

        // Add the angle of rotation and calculate the coordinates after rotation
        Vector2f upperLeftR(centerR.x + distance * cos(angleLeft + rightAngleR * M_PI / 180.f),
            centerR.y + distance * sin(angleLeft + rightAngleR * M_PI / 180.f));
        Vector2f upperRightR(centerR.x + distance * cos(angleRight + rightAngleR * M_PI / 180.f),
            centerR.y + distance * sin(angleRight + rightAngleR * M_PI / 180.f));
        // input these new coordinates into pair
        newPointsR = pair(upperLeftR, upperRightR);

        // Left trees
        pTree(target, newPointsL, depthN-1, leftAngleR);
        // Right trees
        pTree(target, newPointsR, depthN-1, rightAngleR);
    } else {
        return;
    }
}

// Debugger function
```

```
void drawIndicators(sf::RenderTarget& target, pair<Vector2f, Vector2f> newPoints) {
    sf::CircleShape indicatorPoint(5.f);
    indicatorPoint.setFillColor(sf::Color::Red);
    indicatorPoint.setOrigin(5.f, 5.f);
    indicatorPoint.setPosition(newPoints.first);
    target.draw(indicatorPoint);
    indicatorPoint.setFillColor(sf::Color::Green);
    indicatorPoint.setPosition(newPoints.second);
    target.draw(indicatorPoint);
}

// Debugger function
void drawIndicator(sf::RenderTarget& target, Vector2f point) {
    sf::CircleShape indicatorPoint(5.f);
    indicatorPoint.setFillColor(sf::Color::Blue);
    indicatorPoint.setOrigin(5.f, 5.f);
    indicatorPoint.setPosition(point);
    target.draw(indicatorPoint);
}
```