```
#include <fstream>
#include <iostream>
#include <iomanip>
#include <math.h>
#include "console.h"
#include "timer.h"
#include "hashset.h"
#include "lexicon.h"
#include "queue.h"
#include "set.h"
#include "vector.h"
#include "grid.h"
#include "filelib.h"
#include "gwindow.h"
#include "gobjects.h"
#include "simpio.h"
#include "point.h"

using namespace std;

// display constants
static const int SCREEN_WIDTH = 500;
static const int SCREEN_HEIGHT = 500;
static const int LEVEL_HEIGHT = 20;
static const int RECT_HEIGHT = 10;
static const int INSET = 20;

/* Function: Interpolate
 * --------------------
 * Given two points and a fraction (assumed to be in the range
 * [0, 1], the function returns the point "fraction" amount of the
 * way between p1 and p2.
 */
GPoint interpolate(GPoint p1, GPoint p2, double fraction) {
    double deltaX = p2.getX() - p1.getX();
    double deltaY = p2.getY() - p1.getY();
    // if you intrepolate the x and y coords that gives you 2d interpolation
    double x = p1.getX() + fraction * deltaX;
    double y = p1.getY() + fraction * deltaY;
    GPoint newPoint(x, y);
    return newPoint;
}

/* Function: Draw Thick Line
 * --------------------
 * Because sometimes thin lines just don't look good enough in
 * a lecture demo. Actually draws a rectangle instead of a line :)
 */
void drawThickLine(GWindow & w, GPoint left, GPoint right) {
    double width = right.getX() - left.getX();
    // this is called a pointer. We will learn about them later.
    GRect * rect = new GRect(width, RECT_HEIGHT);
    rect->setFilled(true);
    w.add(rect, left.getX(), left.getY() - RECT_HEIGHT/2);
}

/* Function: Get Lowered Point
 * --------------------
 * Returns a GPoint which is LEVEL_HEIGHT pixels lower than the one
 * passed in (has a larger Y value).
 */
GPoint getLoweredPoint(GPoint point) {
    GPoint next(point.getX(), point.getY() + LEVEL_HEIGHT);
    return next;
}

/* Function: Draw Cantor
 * --------------------
 * A recursive function that draws a Cantor Fractal between points
 * "left" and "right." The fractal will have "level" numbers of levels.
 */
```

```
void drawCantor(GWindow & w, int level, GPoint left, GPoint right) {
    if(level > 0) {
        pause(500);
        // 1. draw a single line
        drawThickLine(w, left, right);

        GPoint oneThird = interpolate(left, right, 0.33);
        GPoint twoThird = interpolate(left, right, 0.67);

        // draw the left (smaller) cantor
        drawCantor(w, level - 1, getLoweredPoint(left), getLoweredPoint(oneThird));

        // draw the right (smaller) cantor
        drawCantor(w, level - 1, getLoweredPoint(twoThird), getLoweredPoint(right));
    }
}


/* Function: Main
 * --------------------
 * Draw a Cantor Fractal on the screen.
 */
int main() {
    int depth = 4;

    GWindow w(SCREEN_WIDTH, SCREEN_HEIGHT);
    GPoint left(INSET, INSET);
    GPoint right(SCREEN_WIDTH - INSET, INSET);
    drawCantor(w, depth, left, right);
    return 0;
}
```

```
#include <fstream>
#include <iostream>
#include <iomanip>
#include <math.h>
#include "console.h"
#include "timer.h"
#include "hashset.h"
#include "lexicon.h"
#include "queue.h"
#include "set.h"
#include "vector.h"
#include "grid.h"
#include "filelib.h"
#include "gwindow.h"
#include "gobjects.h"
#include "simpio.h"
#include "point.h"

using namespace std;

// useful math constants
static const double COS_60 = 0.5;            //value of cos(60 degrees)
static const double SIN_60 = sqrt(3)*0.5;    //value of sin(60 degrees)

// display constants
static const int SCREEN_WIDTH = 1000;
static const int SCREEN_HEIGHT = SCREEN_WIDTH;
static const int BASE_Y = SCREEN_HEIGHT - SCREEN_HEIGHT * .4;
static const int BASE_LEFT_X = 170;
static const int BASE_RIGHT_X = SCREEN_WIDTH - 170;

/* Function: Interpolate
 * ---------------------
 * Given two points and a franction (assumed to be in the range
 * [0, 1], the function returns the point "fraction" amount of the
 * way between p1 and p2.
 */
GPoint interpolate(GPoint p1, GPoint p2, double fraction) {
    double deltaX = p2.getX() - p1.getX();
    double deltaY = p2.getY() - p1.getY();
    double x = p1.getX() + fraction * deltaX;
    double y = p1.getY() + fraction * deltaY;
    GPoint newPoint(x, y);
    return newPoint;
}

/* Method: Equilatoral Tip
 * -----------------------
 * Given two points of an equilateral triangle, this method returns the third.
 * It assumes that the first paramenter is the bottom left of the triangle
 */
GPoint equilatoralTip(GPoint p1, GPoint p2) {
    double deltaX = (p2.getX() - p1.getX());
    double deltaY = (p2.getY() - p1.getY());
    double x = p1.getX() + (deltaX*COS_60 + deltaY*SIN_60);
    double y = p1.getY() + (deltaY*COS_60 - deltaX*SIN_60);
    GPoint tip(x, y);
    return tip;
}

/* Method: Draw Flake Line
 * -----------------------
 * Recursively draws a snowflake line (with a given level).
 */
void drawFlakeLine(GWindow & window, int level, GPoint start, GPoint end) {
    // base case
    if (level == 0) {
        window.drawLine(start,end);
    } else {
```

```
        GPoint a = interpolate(start, end, 0.33);
        GPoint b = interpolate(start, end, 0.67);
        GPoint t = equilatoralTip(a, b);
        drawFlakeLine(window, level -1, start, a);
        drawFlakeLine(window, level - 1, a, t);
        drawFlakeLine(window, level - 1, t, b);
        drawFlakeLine(window, level -1, b, end);
    }
}

/* Draw a Snowflake using recursive draw flake line.
 *         Top
 *          X
 *         / \
 *        /   \
 *       /     \
 *      X-------X
 *BottomLeft   BottomRight
 */
int main() {
    int depth = 10;

    GWindow w(SCREEN_WIDTH, SCREEN_HEIGHT);
    GPoint bottomLeft(BASE_LEFT_X, BASE_Y);
    GPoint bottomRight(BASE_RIGHT_X, BASE_Y);
    GPoint top = equilatoralTip(bottomLeft, bottomRight);
    drawFlakeLine(w, depth, bottomLeft, bottomRight);
//    drawFlakeLine(w, depth, bottomLeft, top);
//    drawFlakeLine(w, depth, top, bottomRight);
//    drawFlakeLine(w, depth, bottomRight, bottomLeft);
    return 0;
}
```