

# Assignment 3A: Fractals

Assignment by Marty Stepp and Victoria Kirst. Thanks to Eric Roberts, Julie Zelenski, Jerry Cain, Keith Schwarz for other problem ideas. Updates by Ashley Taylor

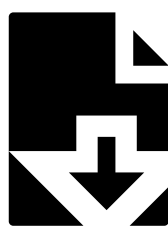
- [Links](#)
- [Description](#)
- [Style](#)
- [Extras](#)
- [FAQ](#)

This problem focuses on recursion.

This is a **pair assignment**. You are allowed to work individually or work with a single partner from your section. If you work as a pair, **comment both members' names** on top of every submitted code file. Only one of you should submit the assignment; do not turn in two copies.

It is fine to write "**helper**" **functions** to assist you in implementing the recursive algorithms for any part of the assignment. Some parts of the assignment essentially *require* a helper to implement them properly. It is up to you to decide which parts should have a helper, what parameter(s) the helpers should accept, and so on. You can declare function prototypes for any such helper functions near the top of your **.cpp** file. (Don't modify the provided **.h** files to add your prototypes; put them in your own **.cpp** file.)

## Links:



Starter Code

We provide a ZIP archive with a starter project that you should download and open with Qt Creator. You will edit and turn in only the following file. The ZIP contains other files/libraries; do not modify these. Your code must work with the other files unmodified. If you want to declare function prototypes, declare them at the top of your **.cpp** file, not by modifying our provided **.h** file.

- **fractals.cpp**, the C++ code for your solution



demo JAR

If you want to further verify the expected behavior of your program, you can download the following provided sample solution demo JAR and run it. If the behavior of our demo in any way conflicts with the information in this spec, you should favor the spec over the demo.

## "How do I run the assignment solution demos?"

If you want to further verify the expected behavior of your program, you can download the provided sample solution demo JAR and run it. If the behavior of our demo in any way conflicts with the information in this spec, you should favor the spec over the demo.

Our assignments offer a solution 'demo' that you can run to see the program's expected behavior. On many machines, all you have to do is download the .jar file, then double-click it to run it. But on some Macs, it won't work; your operating system will say that it doesn't know how to launch the file. If you have that issue, download the file, go to the Downloads folder in your Finder, right-click on the file, and click Open, and then press Confirm.

Some Mac users see a security error such as, "cs106b-hw2-wordladder-demo.jar can't be opened because it is from an unidentified developer." To fix this, go to System Preferences → Security & Privacy. You will see a section about downloaded apps. You should see a prompt asking if you would like to allow access to our solution JAR. Follow the steps and then you should be able to open the demo.

If all else fails, you could run the demo JAR from a terminal. Every operating system allows you to open a "terminal" or "console" window for typing raw system commands. Open your operating system's terminal or console window (Google if you need to learn how to do this), and then type:

```
cd DIRECTORY_WHERE_DEMO_JAR_IS_STORED  
java -jar JAR_FILE_NAME
```

For example, on a Mac machine, if your user name is jsmith12 and you have saved a demo JAR named hw2.jar in your Documents/106b directory, you would type:

```
cd /users/jsmith12/Documents/106b  
java -jar hw2.jar
```

Or on a Windows machine, if your user name is jsmith12 and you have saved a demo JAR named hw2.jar in your Documents/106b directory, you would type:

```
cd C:\users\jsmith12\Documents\106b  
java -jar hw2.jar
```

We provide a GUI for you that helps you run and test your code.

## Problem Description:

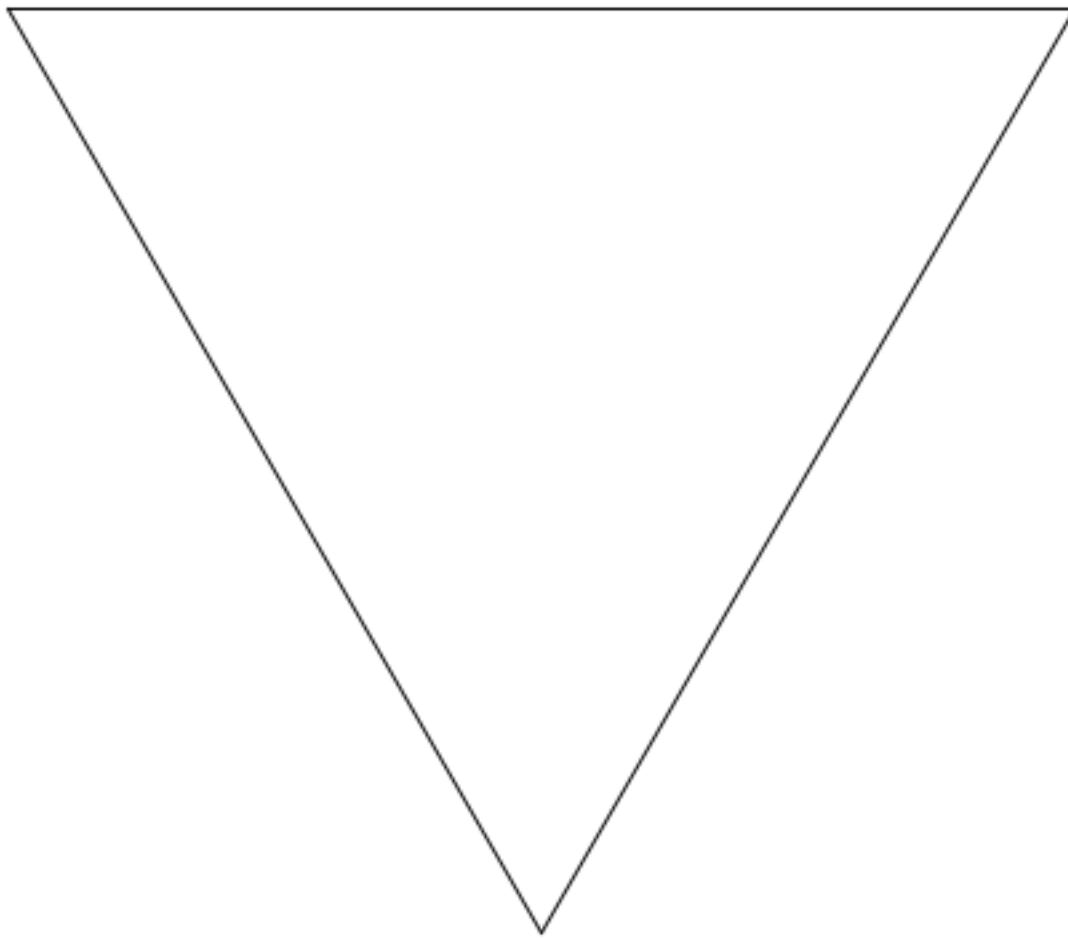
In this problem you will write several recursive functions related to drawing graphics. Recursive graphical patterns are also called *fractals*.

## 1) Sierpinski Triangle

If you search the web for fractal designs, you will find many intricate wonders beyond the Koch snowflake illustrated in Chapter 8. One of these is the Sierpinski Triangle, named after its inventor, the Polish mathematician Waclaw Sierpinski (1882-1969). The order-1 Sierpinski Triangle is an equilateral triangle, as shown in the diagram below.

For this problem, you will write a recursive function that draws the Sierpinski triangle fractal image. Your solution should not use any loops; you must use recursion. Do not use any data structures in your solution such as a **Vector**, **Map**, arrays, etc.

```
void drawSierpinskiTriangle(GWindow& gw, double x, double y, double size, int order)
```



*Order 1 Sierpinski triangle*

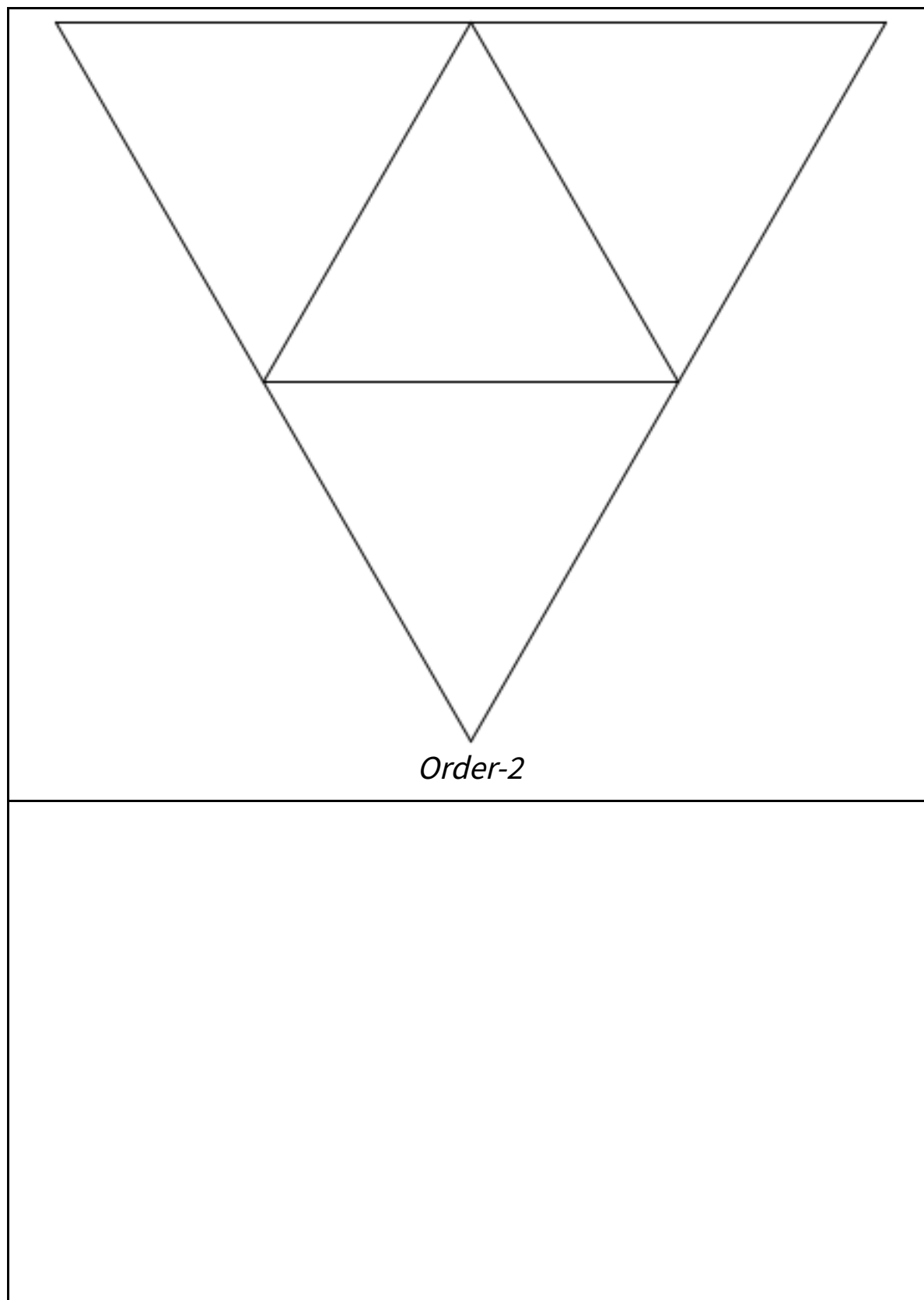
To create an order- $K$  Sierpinski Triangle, you draw three Sierpinski Triangles of order  $K-1$ , each of which has half the edge length of the original. Those three triangles are positioned in what would be the corners of the larger triangle; together they combine to form the larger triangle itself. Take a look at the Order-2 Sierpinski triangle below to get the idea.

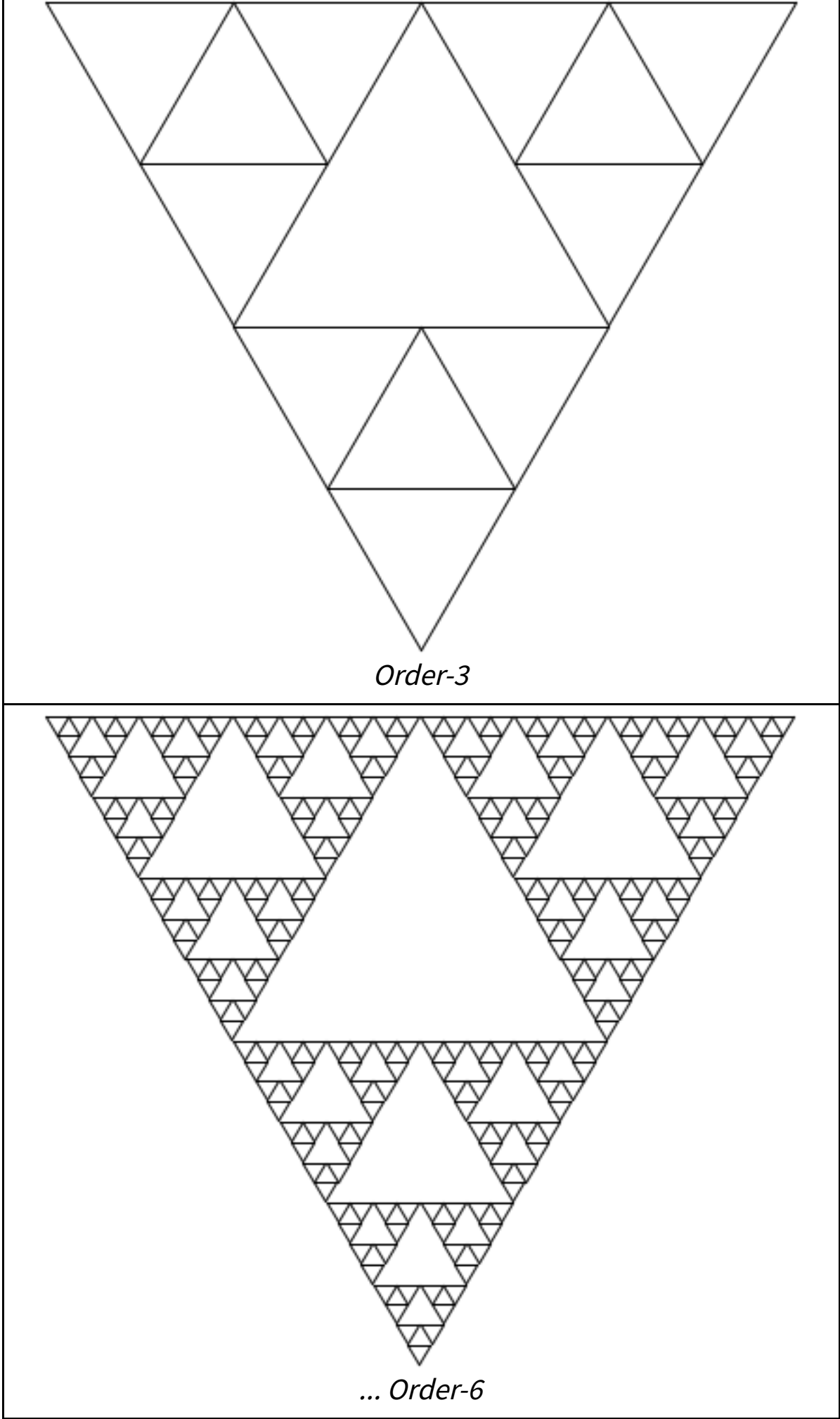
Your function should draw a black outlined Sierpinski triangle when passed a reference to a graphical window, the x/y coordinates of the top/left of the triangle, the length of each side of the triangle, and the order of the figure to draw (such as 1 for Order-1, etc.). The provided code already

contains a **main** function that pops up a graphical user interface to allow the user to choose various x/y positions, sizes, and orders. When the user clicks the appropriate button, the GUI will call your function and pass it the relevant parameters as entered by the user. The rest is up to you.

If the order passed is 0, your function should not draw anything. If the x, y, order, or size passed is negative, your function should throw a string **exception**. Otherwise you may assume that the window passed is large enough to draw the figure at the given position and size.

Some students mistakenly think that some levels of Sierpinski triangles are to be drawn pointing upward and others downward; this is incorrect. The upward-pointing triangle in the middle of the Order-2 figure is not drawn explicitly, but is instead formed by the sides of the other three downward-pointing triangles that are drawn. That area, moreover, is not recursively subdivided and will remain unchanged at every order of the fractal decomposition. Thus, the Order-3 Sierpinski Triangle has the same open area in the middle.



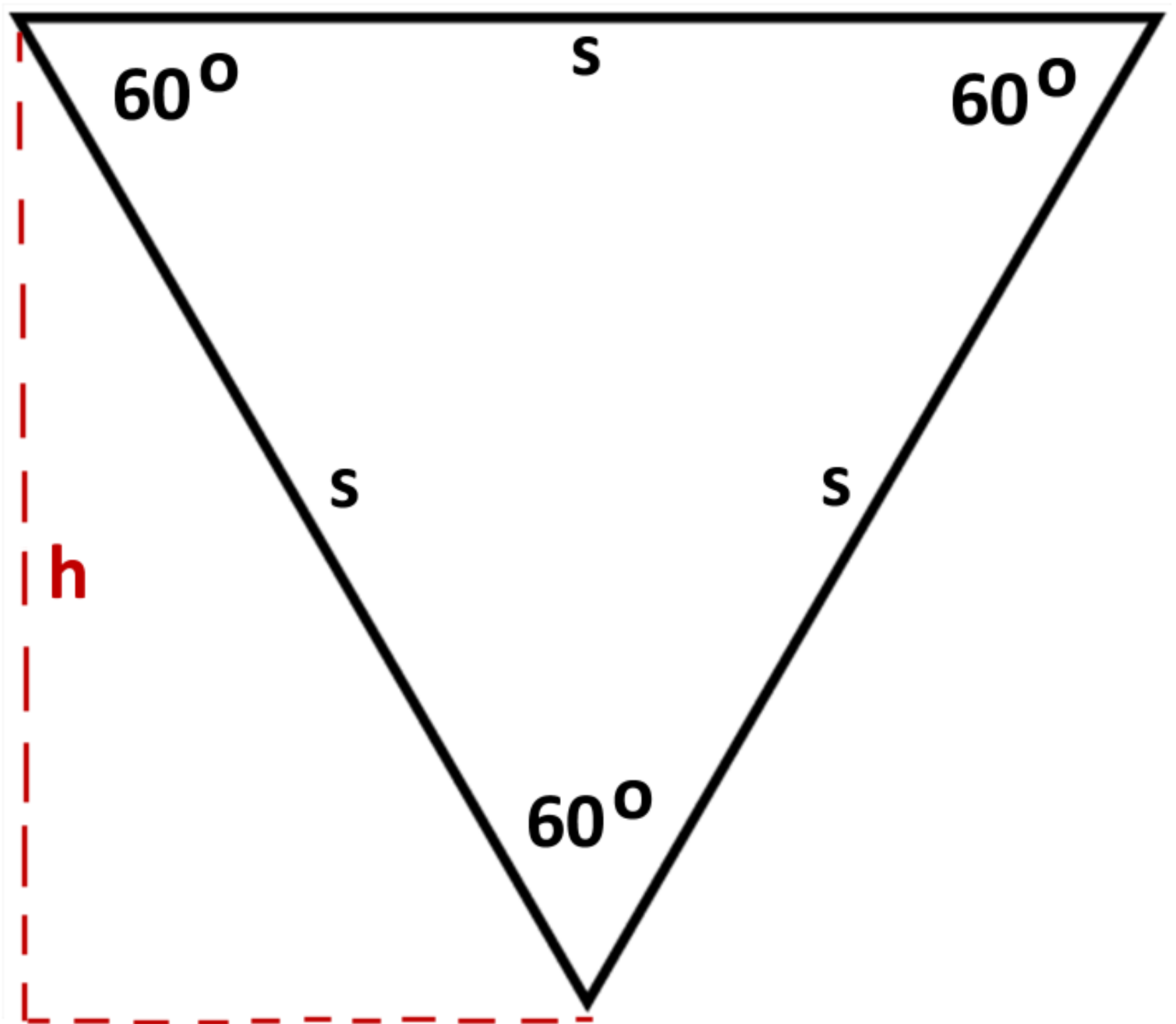


We have not really learned much about the **GWindow** class or drawing graphics, but you do not need to know much about it to solve this problem. The only member function you will need from the **GWindow** is its **drawLine** function. (complete **GWindow** documentation):

Member	Description
<code>gw.drawLine(x1, y1, x2, y2);</code>	draws a line from point (x1, y1) to point (x2, y2)

*Note:* You may find yourself needing to compute the height of a given triangle so you can pass the right x/y coordinates to your function or to the drawing functions. Keep in mind that the height  $h$  of an equilateral triangle is not the same as its side length  $s$ . The diagram below shows the relationship between the triangle and its height. You may want to look at information about equilateral triangles on Wikipedia and/or refresh your trigonometry.

- [http://en.wikipedia.org/wiki/Equilateral\\_triangle](http://en.wikipedia.org/wiki/Equilateral_triangle)



*Computing an equilateral triangle's height from its side length*

A particular style of solution we want you to avoid is the "**pair of functions**" solution, where you write one function to draw "downward-pointing" triangles and another to draw "upward-pointing" triangles, and each one calls the other in an alternating fashion. That is a poor solution that does not capture the self-similarity inherent in this fractal figure.

Another thing you should avoid is **re-drawing** the same line multiple times. If your code is structured poorly, you end up drawing a line again (or part of a line again) that was already drawn, which is unnecessary and inefficient. If you aren't sure whether your solution is redrawing lines, try

making a counter variable that is incremented each time you draw a line and checking its value.

If the order passed is 0, your function should not draw anything. If the x, y, order, or size passed is negative, your function should throw a string **exception**. Otherwise you may assume that the window passed is large enough to draw the figure at the given position and size.

*Expected output:* You can compare your graphical output against the following image files, which are already packed into the starter code and can be compared against by clicking the "compare output" icon in the provided GUI, as shown at right. Please note that due to minor differences in pixel arithmetic, rounding, etc., it is very likely that your output will not perfectly match ours. **It is okay if your image has non-zero numbers of pixel differences from our expected output**, so long as the images look essentially the same to the naked eye when you switch between them.



- sierpinski at x=10, y=30, size=300, order=1
- sierpinski at x=10, y=30, size=300, order=2
- sierpinski at x=10, y=30, size=300, order=3
- sierpinski at x=10, y=30, size=300, order=4
- sierpinski at x=10, y=30, size=300, order=5
- sierpinski at x=10, y=30, size=300, order=6

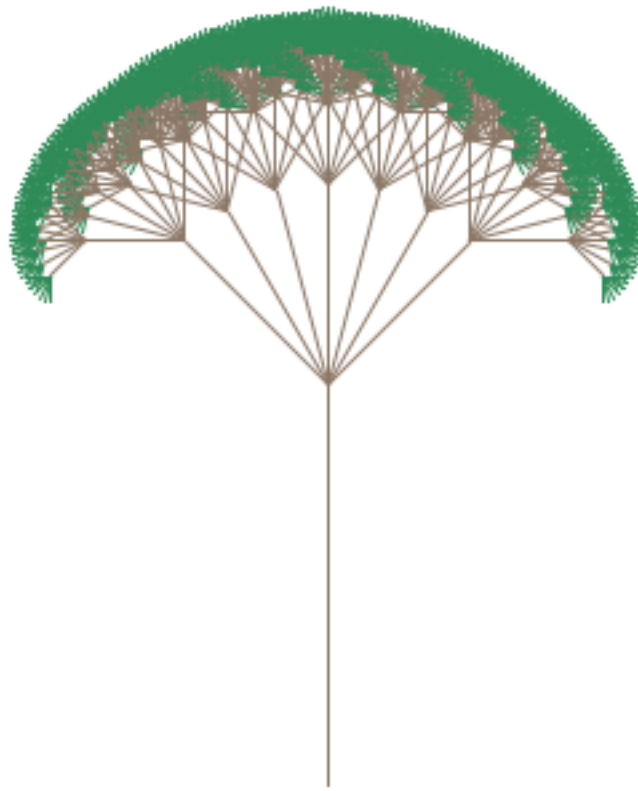
## 2) Recursive Tree

For this problem, write a recursive function that draws a recursive tree fractal image as specified. Your solution is **allowed to use loops** if they are useful to help remove redundancy, but your overall approach to drawing nested levels of the figure must still be recursive. Do not use any data structures in your solution such as a **Vector**, **Map**, arrays, etc.

```
void drawTree(GWindow& gw, double x, double y, double size, int order)
```

Our tree fractal contains a trunk that is drawn from the bottom center of the applicable area ( $x, y$ ) through  $(x + size, y + size)$ . The trunk extends straight up through a distance that is exactly half as many pixels as  $size$ .

The drawing below is a tree of order 5. Sitting on top of its trunk are seven trees of order 4, each with a base trunk length half as long as the prior order. Each of the order-4 trees is topped off with seven order-3 trees, which are themselves comprised of seven order-2 trees, and so on.



*Order-5 tree fractal*

The parameters to your function represent, in order: the window on which to draw the figure; the x/y position of the top/left corner of the imaginary bounding box surrounding the area in which to draw the figure; the general width and height (size) of the figure; and the number of levels (order) of the figure.

Some of these parameters are somewhat unintuitive. For example, the x/y coordinates passed are not the x/y coordinates of the tree trunk itself, but instead the x/y coordinates of a bounding box area in which to draw the tree. The diagram below attempts to clarify this (the GUI is slightly different, but the diagram is still relevant).



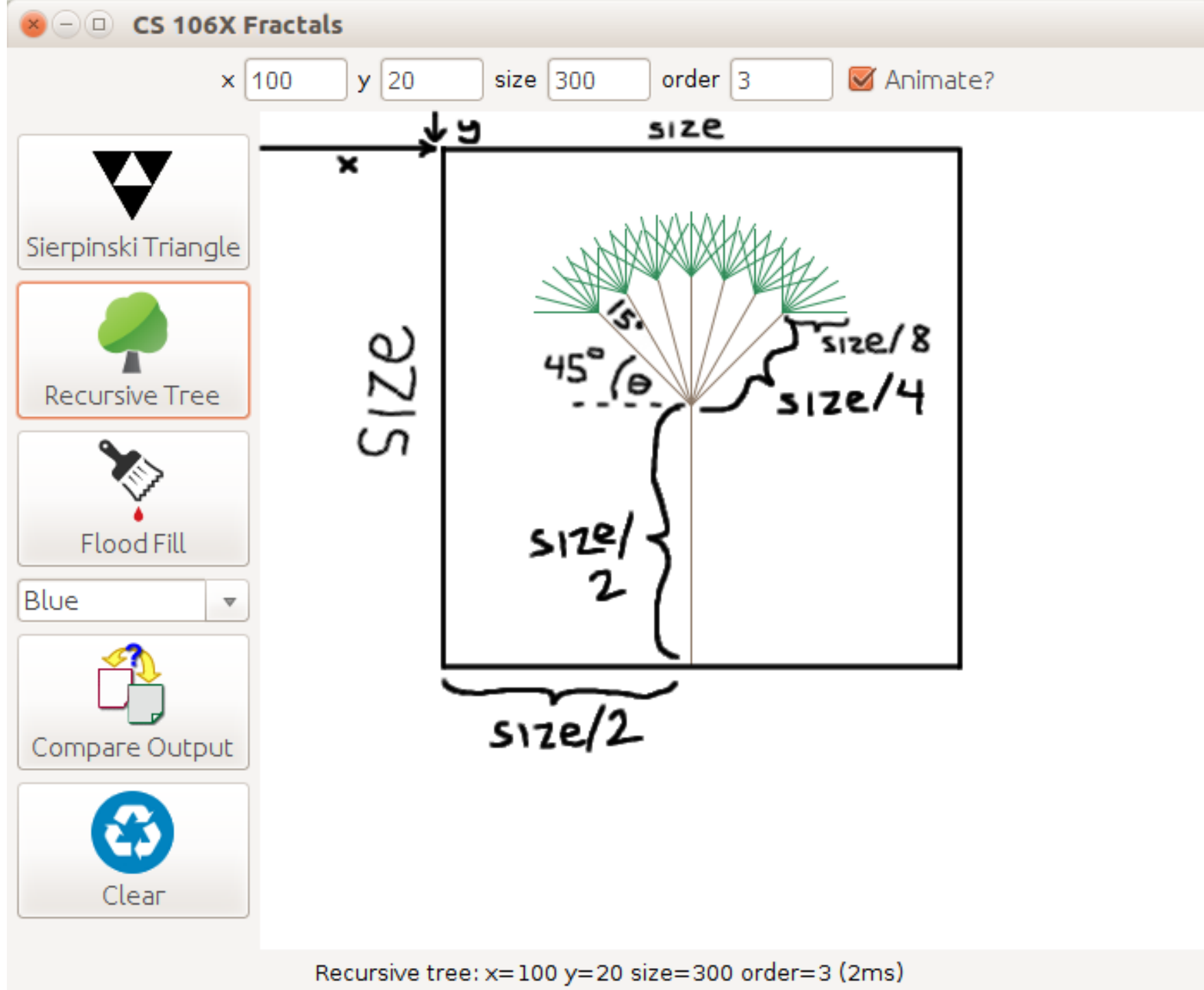


Diagram of `drawTree(gw, 100, 20, 300, 3);` call parameters

**Lengths:** The seven subtrees each have a length that is exactly **half** of their parent branch's length in pixels.

**Angles:** The seven subtrees extend from the tip of the previous tree trunk at relative angles of  $\pm 45^\circ$ ,  $\pm 30^\circ$ ,  $\pm 15^\circ$ , and  $0^\circ$  degrees relative to their parent branch's angle. For example, if you look at the Order-1 figure, you can think of it as a vertical line being drawn in an upward direction and facing upward, which is a polar angle of 90 degrees. In the Order-2 figure, the seven sub-branches extend at angles of 45, 60, 75, 90, 105, 120, and 135 degrees. And so on.

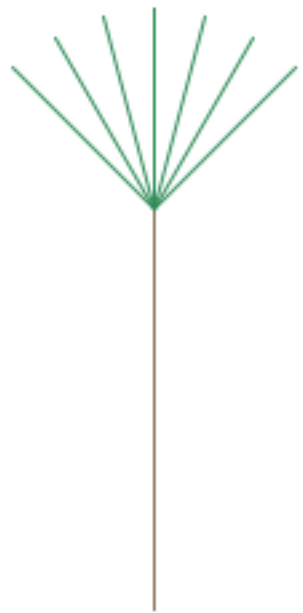
**Colors:** Inner branches, ones drawn at order 2 and higher, are drawn in a color of **#8b7765** (r=139, g=119, b=101), and the leafy fringe branches of the tree (branches drawn at level 1) are drawn in a color of **#2e8b57** (r=46, g=139, b=87).

The images below show the progression of each order of the tree fractal figure.

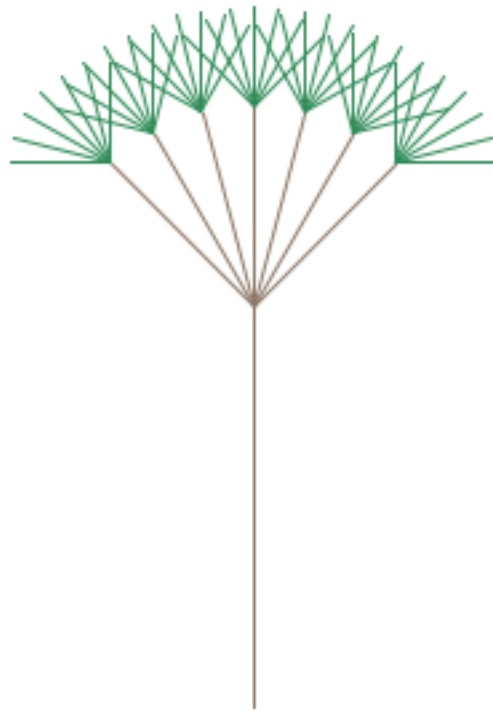




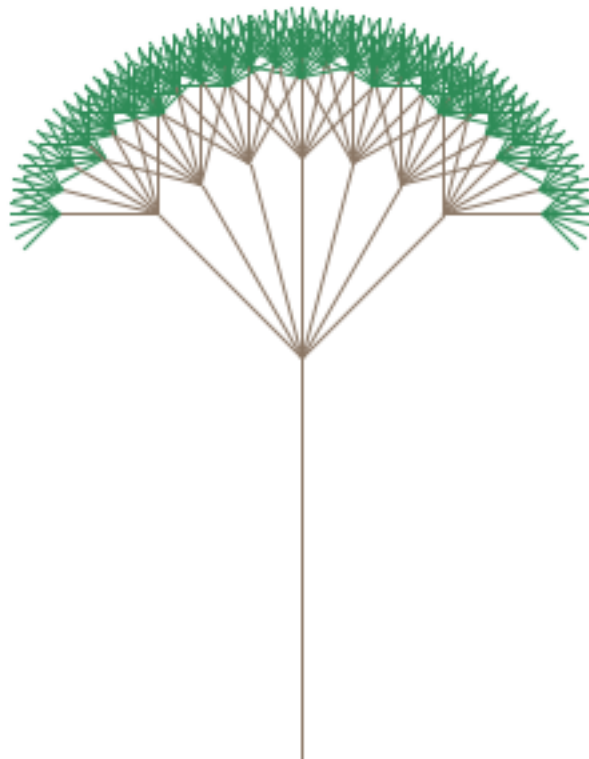
*Order-1*



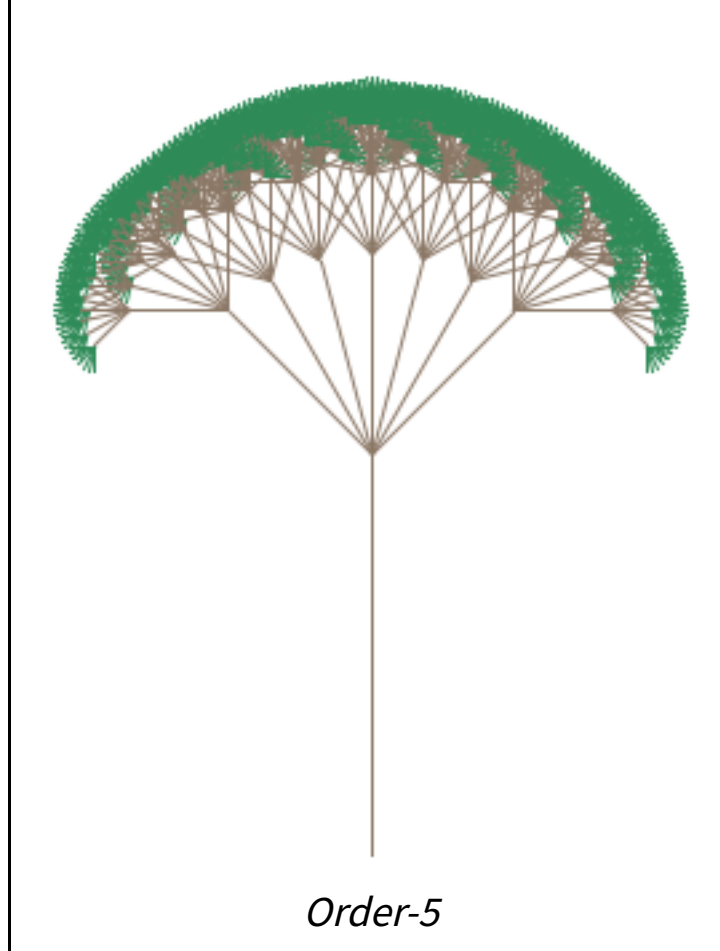
*Order-2*



*Order-3*



*Order-4*



You should use the **GWindow** object's **drawPolarLine** function to draw lines at various angles and its **setColor** function to change drawing colors, as seen in the Koch snowflake example from lecture.

Member	Description
<b><code>gw.drawPolarLine(x0, y0, r, theta);</code></b> <b><code>gw.drawPolarLine(p0, r, theta);</code></b>	draws a line the given starting point, of the given length <i>r</i> , at the given angle in degrees <i>theta</i> relative to the origin. This function returns a <b>GPoint</b> representing the endpoint of the line drawn.
<b><code>gw.setColor(color);</code></b>	sets color used for future shapes to be drawn, either as a hex string such as <b>"#aa00ff"</b> or an RGB integer such as <b>0xaa00ff</b>







If the order passed is 0, your function should not draw anything. If the x, y, order, or size passed is negative, your function should throw a string **exception**. Otherwise you may assume that the window passed is large enough to draw the figure at the given position and size. Note that you can throw a string exception with syntax such as:

```
// throwing a string exception
if (...) {
    throw "your error message text";
}
```

*Expected output:* You can compare your graphical output against the following image files, which are already packed into the starter code and can be compared against by clicking the "compare output" icon in the provided GUI, as shown at right. Please note that due to minor differences in pixel arithmetic, rounding, etc., it is very likely that your output will



not perfectly match ours. **It is okay if your image has non-zero numbers of pixel differences from our expected output**, so long as the images look essentially the same to the naked eye when you switch between them.

-  tree at x=100, y=20, size=300, order=1
-  tree at x=100, y=20, size=300, order=2
-  tree at x=100, y=20, size=300, order=3
-  tree at x=100, y=20, size=300, order=4
-  tree at x=100, y=20, size=300, order=5
-  tree at x=100, y=20, size=300, order=6

## Style Details:

As in other assignments, you should follow our **Style Guide** for information about expected coding style. You are also expected to follow all of the general style constraints emphasized in the Homework 1 and 2 specs, such as the ones about good problem decomposition, parameters, using proper C++ idioms, and commenting. The following are additional points of emphasis and style constraints specific to this problem:

*Recursion:* Part of your grade will come from appropriately utilizing recursion to implement your algorithm as described previously. We will also grade on the elegance of your recursive algorithm; don't create special cases in your recursive code if they are not necessary. Avoid "arm's length" recursion, which is where the true base case is not found and unnecessary code/logic is stuck into the recursive case. **Redundancy** in recursive code is another major grading focus; avoid repeated logic as much as possible. As mentioned previously, it is fine (sometimes necessary) to use "helper" functions to assist you in implementing the recursive algorithms for any part of the assignment.

*Variables:* While not new to this assignment, we want to stress that you should not make any global or static variables (unless they are constants declared with the **const** keyword). Do not use globals as a way of getting around proper recursion and parameter-passing on this assignment.

*Collections:* Do not use any collections on any of the graphical algorithms in this part of the assignment.

## Frequently Asked Questions (FAQ):

For each assignment problem, we receive various frequent student questions. The answers to some of those questions can be found by clicking the link below.

**Q: I'd like to write a helper function and declare a prototype for it, but am I allowed to modify the .h file to do so?**

A: Function prototypes don't have to go in a .h file. Declare them near the top of your .cpp file and it will work fine.

**Q: What does it mean if my program "unexpectedly finishes"?**

A: It probably means that you have infinite recursion, which usually comes when you have not set a proper base case, or when your recursive calls don't pass the right parameters between each other. Try running your program in Debug Mode (F5) and viewing the call stack trace when it crashes to see what line it is on when it crashed.

**Q: The spec says that I need to throw an exception in certain cases. But it looks like the provided main program never goes into those cases; it checks for that case before ever calling my function. Doesn't that mean the exception is useless? Do I actually have to do the exception part?**

A: The exception is not useless, and yes you do need to do it. Understand that your code might be used by other client code other than that which was provided. You have to ensure that your function is robust no matter what parameter values are passed to it. Please follow the spec and throw the exceptions in the cases specified.

**Q: Is the Sierpinski code supposed to be slow when I put in a large, value for N like 15?**

A: Yes; keep in mind that the program is drawing roughly  $3^{15}$  triangles if you do that. That is a lot of triangles. We aren't going to test you on such a high order fractal.

**Q: Are the Sierpinski triangles supposed to exactly overlay each other? Mine seem to be off by ~1 pixel sometimes.**

A: The off-by-one aspect is because the triangle sizes are rounded to the nearest integer. You don't have to worry about that, as long as it is only off by a very small amount such as 1 pixel.

**Q: What does it mean if my program "unexpectedly finishes"?**

A: It might mean that you have infinite recursion, which usually comes when you have not set a proper base case, or when your recursive calls don't pass the right parameters between each other. Try running your program in Debug Mode (F5) and viewing the call stack trace when it crashes to see what line it is on when it crashed. You could also try printing a message at the start of every call to make sure that you don't have infinite recursion.

**Q: Can I use one of the STL containers from the C++ standard library?**

A: No.

**Q: I already know a lot of C/C++ from my previous programming experience. Can I use advanced features, such as pointers, on this assignment?**

A: No; you should limit yourself to using the material that was taught in class so far.

**Q: Can I add any other files to the program? Can I add some classes to the program?**

A: No; you should limit yourself to the files and functions in the spec.

## Possible Extra Features:

Here are some ideas for extra features that you could add to your program for a very small amount of extra credit:

- **Sierpinski colors:** Make your Sierpinski triangle draw different levels in different colors.
- **Add another fractal:** Add another fractal function representing a fractal image you like. You'll have to do some reconstructive surgery on the GUI to achieve this, and/or swap it in place of one of the existing fractal functions (make sure to turn in your non-extra-feature version first, so as not to lose your solution code).
- **Other:** If you have your own creative idea for an extra feature, ask your SL and/or the instructor about it.

*Indicating that you have done extra features:* If you complete any extra features, then in the comment heading on the top of your program, please list all extra features that you worked on and where in the code they can be found (what functions, lines, etc. so that the grader can look at their code easily).

*Submitting a program with extra features:* Since we use automated testing for part of our grading process, it is important that you submit a program that conforms to the preceding spec, even if you want to do extra features. If your feature(s) cause your program to change the output that it produces in such a way that it no longer matches the expected sample output test cases provided, you should submit two versions of your program file: a first one with the standard file name without any extra features added (or with all necessary features disabled or commented out), and a second one whose file name has the suffix **-extra.cpp** with the extra features enabled. Please distinguish them in by explaining which is which in the comment header. Our turnin system saves every submission you make, so if you make multiple submissions we will be able to view all of them; your previously submitted files will not be lost or overwritten.