Program 2

Due Oct 27, 2018 by 11:59pm **Points** 30 **Submitting** a text entry box or a file upload
Available Oct 15, 2018 at 3:30pm - Oct 28, 2018 at 12:05am 12 days

This assignment was locked Oct 28, 2018 at 12:05am.

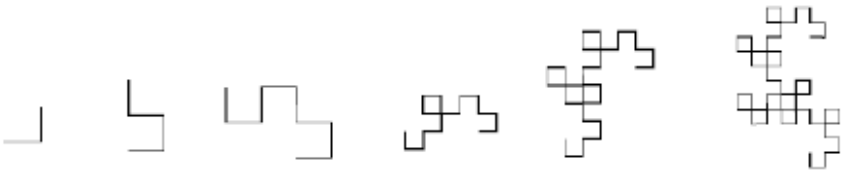
Purpose

This programming assignment draws a complex fractal figure using recursive calls. A "Turtle" class will be provided to draw a line segment or changes its forward direction (Lab 2). Then you should derive a "Dragon" class, which draws **Dragon** curves by calling its methods recursively, using "Turtle"

Please refer the **http://sierra.nmsu.edu/morandi/coursematerials/JurassicParkFractal.html (http://sierra.nmsu.edu/morandi/coursematerials/JurassicParkFractal.html)**

Heighway Dragon Curves

The **Heighway dragon** (also known as the **Harter–Heighway dragon** or the **Jurassic Park dragon**) was first investigated by **NASA** **(http://en.wikipedia.org/wiki/NASA)** physicists John Heighway, Bruce Banks, and William Harter. It was described by **Martin Gardner** **(http://en.wikipedia.org/wiki/Martin_Gardner)** in his **Scientific American** **(http://en.wikipedia.org/wiki/Scientific_American)** column *Mathematical Games* in 1967. Many of its properties were first published by **Chandler Davis (http://en.wikipedia.org/wiki/Chandler_Davis)** and **Donald Knuth** **(http://en.wikipedia.org/wiki/Donald_Knuth)** It appeared on the section title pages of the **Michael Crichton (http://en.wikipedia.org/wiki/Michael_Crichton)** novel ***Jurassic Park (http://en.wikipedia.org/wiki/Jurassic_Park_(novel))***. Here are some examples of Dragon curves.



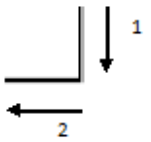
Construction of Dragon

For simplicity, let's focus on 2 levels of dragon curves.

Assuming that Turtle is initially facing right, the following shows the sequence of drawing **a level-1 right Dragon curve**:

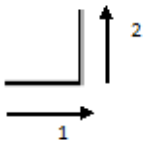
1. Turn -90
2. Draw a straight line

3. Turn -90
4. Draw a straight line



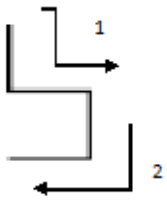
Now, here is **a level-1 curve left curve**;

1. Draw a straight line
2. Turn 90
3. Draw a straight line
4. Turn 90



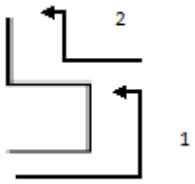
Similar to a level-1 curve, a level-2 curve has two types, left and right. The following sequence draws **a level-2 right Dragon** curve.

1. Turn -90
2. Draw a level-1 left curve
3. Turn -90
4. Draw a level-1 right curve



And a **level-2 left Dragon curve**;

1. Draw a level-1 left curve
2. Turn 90
3. Draw a level-1 right curve
4. Turn 90



Note that right-curve and left curve looks exactly same, but the order of drawing is different.

Now, to go from a level-**k** to a level **k+1**, you need to convert each of line segments of level-1 curve into a level-**k** left or right curve.

Statement of Work

Derive the **Dragon** class from the **Turtle** using class inheritance. You can **You must work on the Dragon class design by yourself**. Note that Dragon is inherited from Turtle:

```
class Dragon : public Turtle
{
...
}
```

This class must include two public methods, **leftCurve** and **rightCurve**, each drawing a level-**k** left and a level-**k** right Dragon curve. Thus, those two methods take two arguments, the depth of level and the length of each line segment. Use the following driver program to verify the correctness of your class design.

```
#include "dragon.h"

#include <iostream>

using namespace std;


#include "dragon.h"
#include <iostream>
using namespace std;

int main( ) {
  // right dragon at level 10
  Dragon dragon10( 80, 360, 0 );
  dragon10.rightCurve( 10, 6 );
  // right dragon at level 9
  Dragon dragon9( 360, 360, 0 );
  dragon9.rightCurve( 9, 6 );

  // right dragon at level 8
  Dragon dragon8( 560, 360, 0 );
  dragon8.rightCurve( 8, 6 );
```

```
  // right dragon at level 7
  Dragon dragon7( 120, 500, 0 );
  dragon7.rightCurve( 7, 8 );

  // right dragon at level 6
  Dragon dragon6( 240, 500, 0 );
  dragon6.rightCurve( 6, 8 );

  // right dragon at level 5
  Dragon dragon5( 360, 500, 0 );
  dragon5.rightCurve( 5, 8 );

  // right dragon at level 4
  Dragon dragon4( 480, 500, 0 );
  dragon4.rightCurve( 4, 8 );

  // right dragon at level 3
  Dragon dragon3( 60, 700, 0 );
  dragon3.rightCurve( 3, 16 );

  // right dragon at level 2

 Dragon dragon2( 240, 700, 0 );

dragon2.rightCurve( 2, 16 );


// right dragon at level 1

  Dragon dragon1( 420, 700, 0 );

  dragon1.rightCurve( 1, 16 );

}
```

To view and print out your result in Linux, follow the directions below:

```
g++ *.cpp
./a.out
  ps2pdf output.ps output.pdf
```

Windows users should view output.ps through **Acrobat**. Mac OS users can view it through **preview**.

Once complete, put the <dragon.h, dragon.cpp, Turtle.h, Turtle.cpp, dragonDriver.cpp> in the same folder into linux

Compile with the following command:

g++ Turtle.cpp dragon.cpp dragonDriver.cpp

What to Turn in

Clearly state in your code comments any other assumptions you have made. Assumptions about main are placed in the beginning comment block of your program. Assumptions about Turtle and Dragon members are placed in their class definition (.h file). (Of course member functions in the .cpp file are also commented.) What you have to turn in includes:

(1) source codes (do not zip)

(1a) dragon.h,

(1b), dragon.cpp,

(option) DragonDriver.cpp (Your own driver if it is different from the one provided)

(2) your execution output for Dragon in **output.pdf** (you need to convert it into PDF).

Note that output must be a PDF file but not **output.ps** file. To get a PDF output on Windows, you must pass **output.ps** to Acrobat Distiller/Reader, generate the correspoonding graphical view, and then save it in PDF.

Grading Guide and Answers

The following is the grading guide how your homework will be graded. Key answers will be made available after the due date at the **Solution** page.

Grade Guideline

```
1. Correctness (26 pts)

Compilation errors(0pt)
Successful compilation without any method implemented (0 pt)

Successful compilation with constructor and at least one recursion method implementation (10 pts)
  + Dragon output pdf files (4 pts)
  + Correct dragon implementation (12 pts)


3. Program Organization (4 pts)
   Proper comments (Have to give comments for each method)
  Good (2pts)       Poor(1 pt)        No explanations(0pt)
```

```
Coding style (proper indentations, blank lines, variable names, and nonredundant code)
Good (2 pts)      Poor(1 pt)        No explanations(0pt)
```