COMP 322: Assignment 3 - Winter 2015

Due at 11:30pm, Mar 22nd 2015

1 Introduction

There are three goals that we would like to attain in this assignment. First, we will write all the mess in the first two homework assignments using classes. Encapsulation should make our lives so much easier with all the methods we dealt with in the previous two assignments. Second, we will illustrate polymorphism by creating a collection of multiple continued fractions. Third, we will generate drawing instructions for generating gifs of interesting growing flowers (this part will be finished in the last homework).

Specific things that you will learn in this assignment:

- Use container classes in the C++ libary: `map`, and `vector`.
- The concepts of encapsulation and polymorphism.
- Use `friend` methods and classes.
- Write instruction files for generating images.

Important reference:

- `http://www.cplusplus.com/reference/stl/map/`
- `http://www.cplusplus.com/doc/tutorial/inheritance/`
- `http://www.imagemagick.org/` : only look at this if you are interested to learn more on the tools we use to draw: `ImageMagick`

NOTE: *Except for the last question, this document does not provide instructions. All instructions are in the header file* `gardens.h`*, as comments attached to each method that you have to implement. This document gives some additional hints and gives you an idea of how we will grade your code.*

NOTE: *Most of the methods you will have to implement here have already been solved in previous assignments. Feel free to use the solution code provided as backbone for your own code. Of course you will have to modify it to use the class definitions we want to obtain, but we don't care if the mathematical and logical details are the same as those we provided.*

2 Encapsulation using classes

In the previous homework assignments we used pointers to deal with continued fractions, and we provided different methods for different types of operations that involved continued fractions. In this homework we will modularize the code from these previous homework assignments by implementing classes. Their definition is provided in the

header file (which includes some declarations as well). Please follow the instructions and guidelines provided as comments over the class definitions in `gardens.h`. Please implement all constructors, the destructor, and member functions for:

Question 1 (0 credits) `ContinuedFraction` - Note that this class has three virtual methods. They are `virtual`: YOU DON'T HAVE TO IMPLEMENT THEM (YET)! Still, you can, and should, use them in `getApproximation`. Make sure to reset the iterator before and after using the iterator methods (if 'iterator' does not make sense to you, read comments in `gardens.h`).

Question 2 (0 credits) `RationalCF` - You can, and should, use the iterator methods in writing the constructor: they are guaranteed to be implemented in `PeriodicCF`. Make sure to reset the iterator before and after using the iterator methods (if 'iterator' does not make sense to you, read comments in `gardens.h`).

Question 3 (25 credits) `PeriodicCF` - This part contains all methods that maintain an iterator over the integers of the continued fraction. Note that in this case, since we have random access to any possible integer in the cf, we keep track of the iterator by using an integer that saves the index the iterator has reached.

Question 4 (25 credits) `MagicBoxCF` - This part contains all methods that maintain an iterator over the integers of the continued fraction. Note that in this case, since we do not have random access to any possible integer in the cf, we need to use the magic box to generate one integer after another. Also, `MagicBox` is a `friend` of `ContinuedFraction`: you have access to all of `ContinuedFraction`'s methods.

3 Friend methods for printing

You will notice that the output operator `operator<<` has been casted as a `friend` of `ContinuedFraction`. The method has access to all its fields.

Question 5 (15 credits) Implement the output operator for objects of type `ContinuedFractions`.

```
ostream &operator<<(ostream &out, const ContinuedFraction &cf);
```

Make sure to return the same output stream `out`, as we would like to chain the operation with other output operations. The format should be the exact same as in previous homework assignments. Feel free to reuse code from the solutions we provided.

4 Sunflowers, e-flowers and other flowers

Reminder from HW2: We want to model the process of growing seeds in flowers. This growth is uniform, all from the center of the flower, pushing all other seeds outward. Mathematically, every time a seed grows, the area of the "flower" grows by 1, and all seeds rotate θ cycles. If we take all seeds from the center towards the outside of the flower, the position of each seed will be:

$$(x_k, y_k) = \left(\sqrt{\frac{k}{\pi}}\cos(2\pi k\theta), \sqrt{\frac{k}{\pi}}\sin(2\pi k\theta)\right)$$

Question 6 (15 credits) Implement the member function that computes the position of the k-th seed, using the formulas above and the solutions to HW2. Do the same with all other constructors, destructor, and member functions for the class `Flower`.

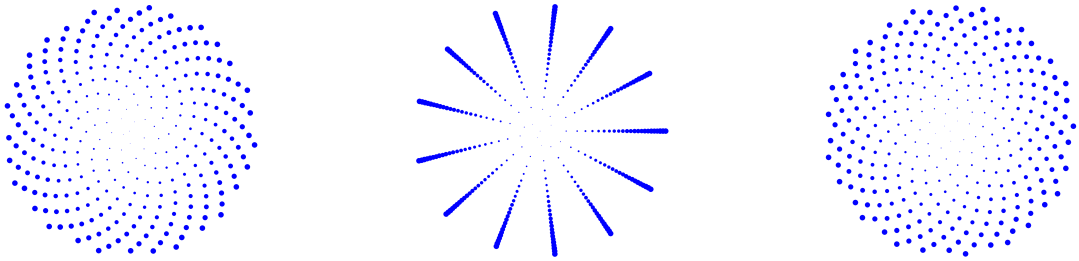

Figure 1: The flowers generated by using square root of 2 (on the left), 30/13 (in the center), and the golden ratio (to the right) to generate the seeds.

See Figure 1 for illustration of flowers that ~~we are looking to obtain in future homework assignments~~ you should be able to generate after finishing the question below. Also, check the course website for a link with more fun facts/math/flowers.

When generating graphics, one has to give the positive `x-y` coordinates, relative to one of the corners, which has coordinates $(0, 0)$. Also, we are drawing pixels, therefore all our coordinates are integers, and they are bounded by the size of the picture/gif/video we want to generate. As such, we need a method that converts the points (x_k, y_k) above, which are relative to the center of the flower, to drawing commands on a canvas with $(0, 0)$ coordinate at the top-left corner. The drawing commands we will use are of the following format:

```
fill blue circle 555,431 565,431
```

The above draws a circle **centered at pixel** with coordinate $(555, 431)$ and with $(565, 431)$ as one if its **border pixels**.

Question 7 (20 credits) Write a method

```
void writeMVGPicture(ostream &, unsigned int N, unsigned int H, unsigned int W) const;
```

That prints such a command for every seed of a flower. This method takes as input the output stream, the number of seeds, and the size of the canvas (i.e. dimensions W and H of the picture to draw). The center pixed for seed (x_k, y_k) should be

$$(C_x, C_y) = \left(\frac{H}{2} + x_k \times \frac{H - 200}{2} \times \sqrt{\frac{\pi}{N}} \ , \ \frac{W}{2} + y_k \times \frac{W - 200}{2} \times \sqrt{\frac{\pi}{N}} \right)$$

where N is the total number of seeds that will be drawn on the canvas. This $H - 200$ madness is there so that the picture has a 100 pixel border that is guaranteed not to be drawn.

The border pixel for the same seed should be

$$(B_x, B_y) = \left(C_x + \sqrt{\frac{k}{N}} \times \frac{\min(W, H)}{100} \ , \ C_y \right)$$

where k is the index of the seed, and N is the total number of seeds that will be drawn on the canvas. This $\sqrt{k/N}$ madness is there so that seeds that are older are drawn larger. The MVG command should be in the format:

```
fill blue circle C_x,C_y B_x,B_y
```

If you save the output of this method to a file, say `dots.mvg`, you can use the following command on the Trottier machines to generate a `.gif` based on these commands:

```
convert -size 1600x1600 dots.mvg dots.gif
```

Where the size should be the same as the W and H used to call the method. `writeMVGPicture`. Please don't submit any such `.gif` file. Generate these pics for your own entertainment.