COMP 322: Assignment 2 - Winter 2015

Due at 11:30pm, Feb 26th 2015

1 Introduction

There are two goals that we would like to attain in this assignment. First, we will learn how to write continued fractions based on the properties of a fraction, as opposed to some predefined periodic representation. Second, we will use continued fractions to start work on generating sunflowers, pi-flowers, e-flowers, and other spiral flowers.

Specific things that you will learn in this assignment:

- Use container classes in the C++ libary: `vector` and `list`. This will provide a brief intro to object oriented programming.
- Use the *algorithms* that come with the C++ library so that you don't need to write everything yourself.

Important reference:

- `http://www.cplusplus.com/reference/stl/vector/`
- `http://www.cplusplus.com/reference/stl/list/`
- `http://www.cplusplus.com/reference/algorithm/`

2 Avoiding pointers and STL data structures

In the previous homework assignment we used pointers to deal with continued fractions. In this homework we will avoid that and use standard libraries that use pointers themselves, but which have been already optimized, tested and widely used. For example, `list` implements the concept of a linked list, `array` implements the concept of a fixed size array, and `vector` implements the concept of a dynamic array (i.e. an array that automatically doubles in size when necessary). For this purpose, a `ContinuedFraction` will be maintained as follows in this new homework:

```
struct ContinuedFraction {
   vector<int> fixedPart;
   vector<int> periodicPart;
}
```

This allows us to easily create the `ContinuedFractions` from the last homework as follows:

```
ContinuedFraction golden, periodic, fixed;
golden.periodicPart = {1};
fixed.fixedPart = {16, 2, 77, 40, 12071};
periodic.fixedPart = {3, 4, 1, 3};
periodic.periodicPart = {1, 2};
```

Suppose we wanted to "spit one integer" out of a continued fraction $[x_0; x_1, x_2, x_3, ...]$–i.e. return x_i, the i-th integer in the continued fraction representation of a number. We would do so this way:

```
int spit(ContinuedFraction &f, int index) {
    auto n = f.fixedPart.size();
    auto m = f.periodicPart.size();
    if(index < n) {
        return f.fixedPart[index];
    }else if (m != 0) {
        return f.periodicPart[(index - n) % m];
    }
    return -1;
}
```

One could write similar methods for other special continued fractions, for example

```
int spitGolden(int index) {
    // golden ratio is [1; 1, 1, 1, ...];
    return 1;
}

int spitSqrt2(int index) {
    // sqrt 2 is [1; 2, 2, 2, ...]
    return index == 0 ? 1 : 2;
}

int spitEuler(int index) {
    // e is [2; 1, 2, 1, 1, 4, 1, 1, 6, ...]
    if(index == 0) return 2;
    if(index % 3 != 2) return 1;
    return (index / 3 + 1) * 2;
}
```

Question 1 (0 credits) Write a method that will modify the `ContinuedFraction` to ignore the integer part. That is, for input $[x_0; x_1, x_2, x_3, ...]$ it returns $[0; x_1, x_2, x_3, ...]$.

Question 2 (0 credits) Google the continued fraction of $\sqrt{8}$ and write a method `int spitSqrt8(int index)` similar to the `spitSqrt2` above.

Question 3 (20 credits) Google the continued fraction of e^2 (the square of the Euler constant) and write a method `int spitEulerSquare(int index)` similar to the `spitEuler` above.

```
int spitEulerSquare(int index);
```

3 Sunflowers, e-flowers and other flowers

We mentioned the golden ratio and the simplicity of its continued fraction in the previous homework. We will now explore the realm of spirals and perfect alignment, which are frequently seen in nature (e.g. sunflower seeds). The latter is the result of seeds growing from the center of the flower at a uniform rate, pushing all other seeds outward. The results is just beautiful! What is the mathematical model of the place of the seeds? We say that

everytime 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, it is not hard to see that the position of each seed will be:

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

Note that $\sqrt{k/\pi}$ is the radial distance the k-th seed has been pushed out (check on your own that the area of a circle of radius $\sqrt{k/\pi}$ is indeed k). Also, $\cos(2\pi k\theta)$ and $\sin(2\pi k\theta)$ compute the x and y components of a point that rotates $k\theta$ cycles.

Question 4 (30 credits) Write a method that computes the angle at which seed k is placed. Note that this number should be between 0 and 2π. To do this, first use Question 1 to get only the fractional part of θ (we only care to get the angle, not the number of rotations), then re-write `getApproximation` form HW1 for the `vector` based representation. Lastly, multiply by k the `Fraction` approximation of θ (get an approximation based on the first 7 integers of `fr`), keep only its fractional part and multiply that by 2π. **Make sure your result is between** 0 **and** 2π.

```
Fraction getApproximation(ContinuedFraction &fr, unsigned int n);
float getAngle(ContinuedFraction &theta, int k);
```

Question 5 (15 credits) Write a function that computes the position of the k-th seed, using the formulas above and Question 4.

```
struct Seed {
   int x, y;
};

Seed getSeed(ContinuedFraction &theta, int k);
```

Question 6 (15 credits) Write a method that will add a new seed to a given flower. Note that each flower is represented by a list of `Seed`s (i.e. their positions relative to the center of the flower).

```
void pushSeed(list<Seed> &flower, ContinuedFraction &fr);
```

See Figure 1 for illustration of flowers that we are looking to obtain in future homework assignments. Also, check the course website for a link with more fun facts/math/flowers.

3.1 Linear transformations of continued fraction

In Question 4 you were asked to multiply a fraction approximation of `fr` by a constant k. We will now learn how to do this without approximations. That is, given two integers a and b, and a continued fraction $x = [x_0; x_1, x_2, ...]$, what is the continued fraction of $f(x) = a + bx$? As expected, this is not very easy, but the algorithm is beautiful (I'll leave the interesting math aside): f becomes a **magical box** that is able to spit the integers in the continued fraction of $f(x) = [y_0; y_1, y_2, ...]$ one after another: y_0, followed by y_1, y_2, etc.

To do this, it maintains four **magical numbers** i, j, k, l that get initialized as $a, b, 1, 0$ respectively, and which change every time one asks for the next integer in the continued fraction.

When asked for the next integer, the following is used as a strategy:

- if both k and l are 0, there are no more integers to spit (i.e. we reached the end of $a + bx$).

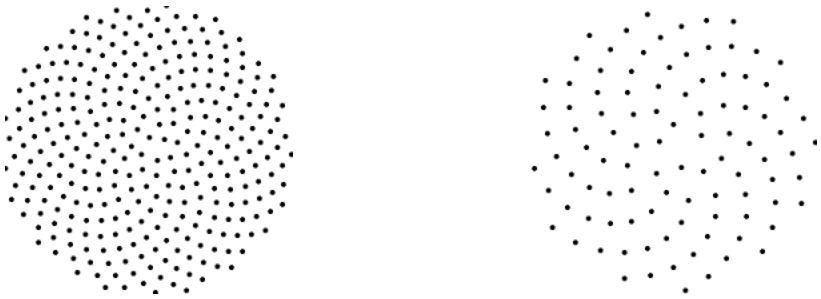

Figure 1: On the left: the flower generated by using the golden ratio to determine the rotation rate. On the right: the flower generated by using $\sqrt{2}$ to determine the rotation rate.

- if i/k and j/l have the same integer part q, it returns this value q AND changes the magical numbers to

$$\left\langle \begin{array}{cc} i & j \\ k & l \end{array} \right\rangle \Rightarrow \left\langle \begin{array}{cc} k & l \\ i-kq & j-lq \end{array} \right\rangle$$

- if i/k and j/l don't have the same integer part q (or if one of k or l are 0, but not both), it changes the box by reading the next integer in the continued fraction of x. Assuming that we did not reach the end of x, we call this next integer p AND change the magical numbers to

$$\left\langle \begin{array}{cc} i & j \\ k & l \end{array} \right\rangle \Rightarrow \left\langle \begin{array}{cc} j & i+jp \\ l & k+lp \end{array} \right\rangle$$

We keep on extracting the next integer in x and change the magic numbers as described above, until the ratios have the same integer part OR until we reach the end of x. When we reach the end of x, the magic numbers get changed to

$$\left\langle \begin{array}{cc} i & j \\ k & l \end{array} \right\rangle \Rightarrow \left\langle \begin{array}{cc} j & j \\ l & l \end{array} \right\rangle$$

At this point the previous case applies and we can extract the next integer in the continued fraction of $f(x)$.

Question 7 (20 credits) Use the following `struct` to implement the algorithm described above for computing linear polynomials of continued fractions

```
struct MagicBox {
ContinuedFraction boxedFraction;
int i = 0, j = 1, k = 1, l = 0;
int indexInBoxedFraction = 0;
}
```

Write a method `getCFUsingMB` that generates the sequence of integers in the continued fraction of $a + bx$, where a, b are integers and x is a continued fraction. This should be done one integer at a time - to make it easier, write first a method `spitNextMagicBox`, which gives the next integer generated by the magic box AND changes the content of the magic box as described above. You should return a `ContinuedFraction` that has a fixed part of length at most l (and no periodic part).

```
int spitNextMagicBox(MagicBox &box);
ContinuedFraction getCFUsingMB(ContinuedFraction &f, int a, int b, int l);
```

Use the answer from Question 2 to check that the Magic Box works well on $f(x) = 0 + 2\sqrt{2}$.