COMP 322: Assignment 4 - Winter 2015

Due at 11:30pm, Apr 14nd 2015

1 Introduction

In this assignment we will wrap up all the work we have done in the previous three assignments, with the focus on three main tasks:

- Use templates to write a class for general fractions.
- Use exceptions to take care of problems that can't be avoided with types and classes.
- Separate the concepts of Continued Fractions and iterators for safer use of multiple iterations over the same continued fraction object.

NOTE: *this document contains partial instructions. Additional information can be found in the header files* `eden.h` *and* `fraction.h`*, as comments attached to each method that you have to implement. This document provides 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.*

Question 1 (40 credits) You should implement a template class for fractions, as we have been using in the previous and current assignment for dealing with continued fractions. Please read the header file `fraction.h` and complete all methods that are listed as `//TODO`. We will allow division by 0 (thinking of it as infinity), but 0/0 will be disallowed, as it has no meaning whatsoever. That is,

- Infinity is always 1/0. If the result of any operation results in denominator 0, the numerator should be changed to 1.
- Zero is always 0/1. If the result of any operation results in numerator 0, the denominator should be changed to 1.
- If one wants to create a fraction with both numerator and denominator equal to 0, an exception of type `int` equal to 0 should be thrown.
- If the result of any operation is 0/0, an exception of type `int` equal to 0 should be thrown. Make sure that methods not declared as `const` will NOT change the value of the fraction unless the result is different from 0/0.

Write your code for this question in `fraction.h`.

Question 2 (0 credits) Implement the two methods below. Notice that now we are using the template class `Fraction`, which does not allow us to change the numerator and the denominator at wish. You will have to use fraction arithmetic this time:

```
Fraction<cf_int> ContinuedFraction::getApproximation(unsigned int k) const;
RationalCF::RationalCF(Fraction<cf_int> f);
```

For details on how these should work, see the header file `eden.h`.

Note: The following exceptions should be thrown:

- `getApproximation` should throw an `int` equal to 0 if the parameter `k` is equal to 0.
- the constructor `RationalCF` should throw an `int` equal to 0 if the fraction `f` is infinity (i.e denominator is 0).

Write your code for this question in `edenStudent.cpp`.

Question 3 (20 credits) To make sure that we access the data of each continued fraction in a safe way, we will implement a class `Iterator` which will allow us to create iterator objects. This is different than the previous homework, in which we only had one object which was maintaining both the data AND the iterator. Some of you cleverly noted that that design was not very clever.

For an initial practice, implement an iterator class for the continued fraction of e^2. Feel free to use the solution we gave for e^2 in Homework 2.

Note: After you finish implementing the methods in `EulerSquaredIterator`, make sure you implement

```
Iterator *EulerSquared::getIterator() const;
```

which returns a pointer to a `EulerSquaredIterator` object.

Write your code for this question in `edenStudent.cpp`.

Question 4 (20 credits) Much like in Question 3, implement the iterator class for periodic continued fractions, and the method

```
Iterator *PeriodicCF::getIterator() const;
```

which returns a pointer to a `PeriodicCFIterator` object.

Note: You should throw an exception of type `double` equal to `-1.0` if one calls `next()` when the iterator `isDone()`.

Write your code for this question in `edenStudent.cpp`.

Question 5 (20 credits) Much like in Question 3, implement the iterator class for Magic Box continued fractions, and the method

```
Iterator *MagicBoxCF::getIterator() const;
```

which returns a pointer to a `MagicBoxCFIterator` object.

Note: You should throw an exception of type `double` equal to `-1.0` if one calls `next()` when the iterator `isDone()`.

Write your code for this question in `edenStudent.cpp`.