Lab 6:
Array, a Drop of Golden Sun

Due date: March 24

Overview

You will once again reuse your `Rational` class for this lab. In this assignment, you will create a class to represent a **dynamic-sized** array of `Rational` objects. This `RationalArray` class will have a size determined at run-time, and will overload operators to let you index a `RationalArray` object in order to access its underlying array of `Rational` objects. It will properly manage any memory it places onto the heap, deleting variables when they are no longer needed through destructors and proper deep copy constructors/assignment operators.

Assignment

Start a new project and copy your `Rational.h` and `Rational.cpp` files there. Download the file `RationalArray.h` from the course website and add it to the project. Read through the file to make sure you understand everything inside it – you will be implementing functions from that file.

A `RationalArray` object contains two member variables: an integer size to remember the array's length, and a pointer to a dynamically-sized built-in array on the heap. When a RationalArray is constructed (requiring a size), you will allocate an array of the requested size on the heap and save the pointer given to you by `new` into the member variable in the `RationalArray` object. You will then use that member pointer to perform indexing operators, and will properly delete it when it is no longer needed.

Implement the following functions in `RationalArray.cpp`:

1. Constructors:
   (a) `Rational(int size)`: initialize the size of the array, and allocate a new array of `Rational` objects on the heap of the given size.
   (b) `Rational(const Rational &other)`: perform a **deep copy** of `other`, so that any subsequent change to `other` will not change the newly-constructed array.
2. A **destructor** which cleans up any memory allocated on the heap by the RationalArray object.
3. An **assignment** `operator=`, which performs a **deep assignment** of an existing `RationalArray` into another existing `RationalArray`. Since the `lhs` array already exists, you will need to be careful that any heap space belonging to `lhs` gets freed before overriding its array pointer.
4. An `operator[]`, which you may need to look up, which returns the `Rational` object at the specified array index **by reference**. This method should **throw** `std::out_of_range` if the specified index is not in bounds of the array.

Test Program

When you are ready to test your code, download the file `RationalTester.cpp` from BeachBoard and add it to your project. "Run" the code in `main` by hand to predict what the outcome should be, given your knowledge of constructors, deep vs shallow copies, and operators. **If the main does not compile, you have not followed the directions of the last three labs – make sure your methods and operators are implemented as required**.

After the tests complete to your satisfaction, **complete the required program at the end of the main**. Your program will find and output the **mean (average)** of all the `Rational` objects in the array constructed by the main. You should not have to do anything particularly tricky to do this calculation, **if you correctly**

implemented `operator+` and `operator/`. (You may need a minor trick to get the final average through division.)

Deliverables

Hand in:

1. A printed copy of your code, **printed from Visual Studio or your IDE when possible.** If you cannot print from your editor, copy your code into Notepad or another program with a fixed-width (monospace) font and print from there. Print **all .h and .cpp files**.
2. The printed output of your code.