CSCI 15 Assignment #4, introduction to classes/objects. 100 points, due 10/23/18

A mixed expression is an expression of the form a + b/c, where a, b, and c are integers, b >= 0 and c > 0. By convention, b/c is always non-negative, and has the property that 0 <= (b/c) < 1. Also, b/c is always shown in reduced form, or as 0 / 1.

Implement the `MixedExpression` class, allowing use of the mixed expression as an abstract data type. Also, write a calculator client to drive the `MixedExpression` class. Your `MixedExpression` class must be described in a header file (`mixedexpression.h`) and implemented in a separate library source file (`mixedexpression.cpp`), both of which are in the same working directory as the client calculator program. The client code will `#include` the header you wrote, and will be linked with the class library object file. Create a project for the 2 source files.

Your `MixedExpression` class must provide public member functions (methods) to add, subtract, multiply and divide mixed expressions. These must have the form:

```
MixedExpression MixedExpression::add( MixedExpression op )
```

This will add "this" mixed expression (e.g., "self" – the addend) to the argument `op` , the augend (remember, addend + augend yields sum) and return the sum. The other methods; `subtract()`, `multiply()`, and `divide()`, will also take a mixed expression (the argument) and treat it as the right hand operand of their appropriate operation, using self (the object to the left of the '`.`') as the left hand side operand. All of these functions will create a new MixedExpression, set its values, and return it so it can be assigned to the variable holding the result.

Given MixedExpression variables res, op1 and op2, the call in the calculator code will look like this:

```
res = op1.add( op2 );
```

You must also provide appropriate constructors, observers (i.e., to print a mixed expression, in the form (a + b / c), where parentheses and spaces are required), and manipulators (i.e., to take supplied values and set them into the mixed expression).

Your class data members (`long a, b, c;`) must be private.

YOU MAY NOT ADD ANY OTHER DATA MEMBERS TO THE CLASS.

YOU MAY NOT ADD ANY OTHER PARAMETERS TO THE METHODS.

You may need private "helper" functions, for example, a private method `reduce()` to convert a mixed expression to its "normal" form.

Your program will read lines of text which are in the form

```
( a + b / c ) <operator> ( d + e / f )
```

where the parentheses `( )`, the `+`, and the `/` are literals, `<operator>` ∈ {`+`, `-`, `*`, `/`}, and `a`, `b`, `c`, `d`, `e` and `f` are signed integers. Do not try to read and parse the lines yourself: your `MixedExpression` objects will read the operands themselves (as described below), and you will extract the operator directly inside your calculator.

Your input method, `ReadMixedExp( istream &in )` is a method of the class `MixedExpression`, and extracts the object's data values from input. Your client will call this twice per line of input, once for each operand. The parameter type `istream &` means the method can take either an `istream` or an `ifstream` as an argument. I will explain how this works later.

Your client must extract the operator itself (a mixed expression doesn't understand math operators!) between extracting the operands. The client must interpret the operator itself.

You may assume the input is correctly formatted. Use an input file and process one line at a time until you reach end-of-file. Output is to a text file. Pass the file names into the program via the command line. Print the entered line (converted to reduced normal form if needed), followed by =, then the result.

The operands and the result must print themselves. That is, you must use a method `printData( ostream &out )` {which can take either an `ostream` or an `ofstream`} to print the mixed expression.

For example, with this input line

```
( 1 + 2 / 6 ) + ( 3 + -3 / 9 )
```

The corresponding output would be (notice changes from the input form to reduced normal form)

```
( 1 + 1 / 3 ) + ( 2 + 2 / 3 ) = ( 4 + 0 / 1 )
```

This spacing and format of the input and output lines is required.

Test your program with several sets of data, including attempts at illegal data or operations, such as

```
( 1 + 0 / 1 ) / ( 0 + 0 / 1 )
```

Your program must catch and appropriately handle undefined expressions such as 1+1/0. Just use the value 0+0/1 in these cases. Later, we'll see how to really handle these errors.

Do not try to put the calculator into the class! The class just represents a single data object (variable) at a time: you will need three of these objects to hold the two operands and the result.