**Logo

Description automatically generated San Francisco Bay University**

**CS360L - Programming in C and C++ Lab**

**Lab Assignment #4**

**Due day: 6/14/2022**

**Instruction:**

1. **Push the answer sheets/source code to Github**
2. **Please follow the code style rule like programs on handout.**
3. **Overdue lab assignment submission can’t be accepted.**

**4. Take academic honesty and integrity seriously (Zero Tolerance of Cheating & Plagiarism)**

1. Correct the errors in the following snippets and explain
   1. Assume the following prototype is declared in class *Time*:

*void ~Time( int );*

* 1. Assume the following prototype is declared in class *Employee*:

*int Employee( string, string );*

* 1. The following is a definition of class *Example*:

*class Example{*

*public:*

*Example( int y = 10 ): data( y ){*

*// empty body*

*} // end Example constructor*

*int getIncrementedData() const{*

*return ++data;*

*} // end function getIncrementedData*

*static int getCount(){*

*cout << "Data is " << data << endl;*

*return count;*

*} // end function getCount*

*private:*

*int data;*

*static int count;*

*}; // end class Example*

1. Generate a class called *Rational* to perform arithmetic with fractions. Write a program to test your class. Use integer variables to represent the private data of the class--the *numerator* and the *denominator*. Provide a constructor that enables an object of this class to be initialized when it’s declared. The constructor should contain default values in case no initializers are provided and should store the fraction in reduced form. For example, , the fraction would be stored in the object as *1* in the numerator and *2* in the denominator. Provide public member functions that perform each of the following tasks:
   1. Adding two *Rational* numbers. The result should be stored in reduced form.
   2. Subtracting two *Rational* numbers. The result should be stored in reduced form.
   3. Multiplying two *Rational* numbers. The result should be stored in reduced form.
   4. Dividing two *Rational* numbers. The result should be stored in reduced form.
   5. Printing *Rational* numbers in the form *a/b*, where *a* is the numerator and *b* is the denominator.
   6. Printing *Rational* numbers in floating-point format.
2. Create a class *HugeInteger* that uses a *40*-element array of digits to store integers as large as *40* digits each. Provide member functions *input, output, add* and *subtract*.For comparing *HugeInteger* objects, provide functions *isEqualTo, isNotEqualTo, isGreaterThan, isLessThan, isGreaterThanOrEqualTo* and *isLessThanOrEqualTo*--each of these is a "predicate " function that simply returns *true* if the relationship holds between the two *HugeIntegers* and returns *false* if the relationship does not hold. Also, provide a predicate function *isZero*. After that, provide member functions *multiply, divide* and *modulus*.
3. Create a *SavingsAccount* class. Use a *static* data member *annual-InterestRate* to store the annual interest rate for each of the savers. Each member of the class contains a *private* data member *savingsBalance* indicating the amount the saver currently has on deposit. Provide member function *calculateMonthlyInterest* that calculates the monthly interest by multiplying the *savingsBalance* by *annualInterestRate* divided by *12*; this interest should be added to *savingsBalance*. Provide a *static* member function *modifyInterestRate* that sets the *static* *annualInterestRate* to a new value. Write a driver program to test class *SavingsAccount*. Instantiate two different objects of class *SavingsAccount*, *saver1* and *saver2*, with balances of *$2000.00* and *$3000.00*, respectively. Set the *annualInterestRate* to *3* percent. Then calculate the monthly interest and print the new balances for each of the savers. Then set the *annualInterestRate* to *4* percent, calculate the next month's interest and print the new balances for each of the savers.