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

```
void ~Time( int );
```

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

```
int Employee( string, string );
```

    c. 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
```

The provided snippet seems to be an attempt to declare the destructor for the class "Time". However, the destructor in C++ should use the tilde (~) character followed by the class name and should not have any return type. The correct syntax for the destructor declaration should be:

```
~Time();
```

The provided snippet seems to be an attempt to declare a constructor for the class "Employee". In C++, the constructor name should be the same as the class name and should not have any return type. The correct syntax for the constructor declaration should be:

```
Employee( string, string );
```

The provided class "Example" contains a few errors

In the constructor definition, the default argument "y = 10" should be outside the parentheses. The correct definition for the constructor should be:

```
Example( int y = 10 ): data( y ) {}
```

The corrected class "Example" with the mentioned errors fixed is:

```
#include <iostream>

class Example {
public:
    Example(int y = 10): data(y) {}

    int getIncrementedData() const {
        return ++data;
    }

    static int getCount() {
        std::cout << "Count is " << count << std::endl;
        return count;
    }

private:
    int data;
    static int count;
};

int Example::count = 0; // Initialization of the static member "count"
```

2. 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, $\frac{2}{4}$, 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:

    a. Adding two *Rational* numbers. The result should be stored in reduced form.
    b. Subtracting two *Rational* numbers. The result should be stored in reduced form.
    c. Multiplying two *Rational* numbers. The result should be stored in reduced form.
    d. Dividing two *Rational* numbers. The result should be stored in reduced form.
    e. Printing *Rational* numbers in the form *a/b*, where *a* is the numerator and *b* is the denominator.
    f. Printing *Rational* numbers in floating-point format.

Here's the output and the main code is in the cpp file

```
Sum: 11/10 or 1.1
Difference: 1/-10 or -0.1
Product: 3/10 or 0.3
Quotient: 5/6 or 0.833333
```

3. 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*.

Here's the output and the main code is in the cpp file

```
Sum: 11111111011111111101111111110111111111100
Difference: 13580246791358024679135802467913580246 80
Product: 958695313578722742498094791124980947000
Modulus: 12345678901234567890123456789012345678 90


...Program finished with exit code 0
Press ENTER to exit console.
```

4. 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.

Here's the output and the main code is in the cpp file

```
Initial balances:
Saver 1: $2000
Saver 2: $3000

Balances after 1 month at 3% interest rate:
Saver 1: $2005
Saver 2: $3007.5

Balances after 1 more month at 4% interest rate:
Saver 1: $2011.68
Saver 2: $3017.53
```