Work on the following exercises in the sequence indicated.

**Logging On.** Log on with your username and password. If you experience any difficulty, let the lab instructor know immediately. Insist that your problem be fixed in the beginning of this lab.

The topic of this lab is definition of user-defined **classes**, and the programming with **objects**. You may think of object simply as variables that are of a type that happens to be a user-defined class. In this lab you will define a class **Fraction** which will allow us to represent numeric values as fractions, e.g., $\frac{1}{2}$, $\frac{5}{3}$, $\frac{7}{21}$, etc.

**Exercise 1: A simple class Fraction.** Open a new file **fraction.cpp** and define a class **Fraction** as suggested by the following “skeleton” class:

```cpp
class Fraction
{
  public:
    // constructor that initializes numerator and denominator;
    // allow only positive fractions, and do not allow 0 for
    // the denominator;

    Fraction(int n, int d)
    {
      // ...
    }

    // set function: sets numerator and denominator to values passed;
    // allow only positive fractions, and do not allow 0 for
    // the denominator;

    void set(int n, int d)
    {
      // ...
    }

    // get function: returns value of the numerator;
    int get_numer()
    {
      // ...
    }
};
```

// get function: returns values of the denominator;
int get_denom()
{
  // ...
}

// prints the fraction in form of x/y;
void print()
{
  // ...
}

private:

  // two integer data members, one for numerator, one for
  // denominator;
  // ...

};
Test your class with the following `int main()` function:

```cpp
int main()
{
    int a, b;

    cout << endl;
    cout << "Enter the numerator and denominator of your first fraction: ";
    cin >> a >> b;

    Fraction frac1(a,b);

    cout << endl;
    cout << "Enter the numerator and denominator of your second fraction: ";
    cin >> a >> b;

    Fraction frac2(a,b);

    cout << endl;
    cout << "Your fractions are: ";
    frac1.print();
    cout << " and ";
    frac2.print();
    cout << endl << endl;

    return 0;
}
```

Do not forget all necessary `#include` statements and `using namespace std`; Compile and test for a variety of inputs.

**Exercise 2.: Normalizing fractions.** Enhance your class `Fraction` by adding the capability to convert a fraction into its *normalized* form. A fraction is normalized when its numerator and denominator have taken on the smallest possible integer values that, as fraction, represent the same value as the original fraction. For example, fraction $\frac{3}{15}$ is normalized to $\frac{1}{5}$. In general, a fraction is normalized by dividing the numerator and denominator by their *greatest common factor* (which is 3 for the example).

The following recursive function is the quickest way to write a function that will, for any two integers, determine the greatest common factor.
// greatest common factor;

int gcf(int a, int b)
{
    if (a % b == 0)
        return b;
    else
        return gcf(b, a % b);
}

Since we have not (yet) formally discussed recursion in the lecture, you may simply take this function for granted. Add this function to your file fraction.cpp. Make sure that either the prototype, or the entire function definition appears before class Fraction.

Within class Fraction add a member function

void normalize()
{
    // ...
}

with its body implementing the normalization. In order to test your new member function, add the following lines to the end of your int main() in Exercise 1:

    frac1.normalize();
    frac2.normalize();
    cout << "They are normalized: ";
    frac1.print();
    cout << " and ";
    frac2.print();
    cout << endl << endl;

Compile and test with a few example inputs. Verify that your fractions are being normalized, and realize that some fractions may already be in this form from the start (and normalizing will not change them).

Exercise 3.: Adding fractions. Enhance your class Fraction even further by adding a member function that allows the addition of two Fraction objects. Given two fractions, frac1 and frac2, you should be able to add both with a member function call

    Fraction sum = frac1.add(frac2);
Notice how frac1 is plays the role of the calling object, and frac2 is the argument object of the .add member function call.

Recall the mathematical operations to add two fractions. The following steps are easiest to implement:

\[
\frac{x \cdot z + w \cdot y}{x/y + w/z} = \text{NORMALIZE}\left(\frac{x \cdot z + w \cdot y}{y \cdot z}\right)
\]

Add a member function add to class Fraction:

```cpp
Fraction add(Fraction other) {
    // ...
}
```

Test the new member function by adding to your int main() the following lines of code:

```cpp
Fraction sum = frac1.add(frac2);
cout << "Result of adding both: ";
sum.print();
cout << endl;
```

Compile and test with a few examples.