Chapter 14
Friends of Classes

- Reading pp.819-823, 831-857
- Good Problems to work: p. 823 14.1, 14.2, 14.3, 14.4, 14.5, 14.6, 14.7; p. 845 14.14, 14.19; p. 857 14.21

Friends of Classes

- Only the member functions of a class have direct access to the private data members of the class
- `friend` functions are friends of the class that are defined outside of the class but still have access to private data members

Friends of Classes

- A friend function can be
  a) a regular stand-alone function
  b) a member of another class
  c) an entire class

friend Functions

- The function prototype is placed in the class, preceded by the keyword friend
- The function definition can be written anywhere without the class name (class::)
- The function is able to directly access the private data members of the class

Operator Overloading

- C++ allows overloading operators to work with class objects
- For example, we can overload the extraction operator >> and insertion operator << for Rational objects

Overloading Stream Operators

- Two classes named ostream and istream provide stream I/O.
- Definitions for >> and << are provided for the primitive datatypes such as int, float, char, and so on but not for programmer-defined types. Stream operators can be overloaded for our **Rational** class. As an example, we would like the following to have meaning:

```
Rational cR1;

std::cout << "Enter a rational number:";
std::cin >> cR1;
```

In particular, we would like to be able to enter a value such as `1/3` for cR1.

Overloading Stream Operators

- The general format for overloading the stream operators is as follows:

```
class SomeClassName
{
  public:
    .....
    friend std::istream& operator>> (std::istream &rcInput,
                                     ClassDef &rcObject);
    friend std::ostream& operator<< (std::ostream &rcOutput,
                                     const ClassDef &rcObject);
  private:
    .....
};
```

Overloading Stream Operators

- Note: For the stream extraction operator >> some istream object is passed to the operator function through **cInput** such as cin.
- Similarly, the stream insertion operator << is passed some ostream object through **cOutput** such as cout.
- The function returns a modified stream so that the following chain can be executed correctly:
- `std::cin >> r1 >> r2;` // What is the associativity of >>? Why?
- Similar logic is used for the insertion operator << function.

Overload Insertion Operator <<

```
class Rational
{
  public:
    Rational (int numerator = 0, int denominator = 1);
    .....
    friend std::ostream& operator<< (std::ostream &rcOutput,
                                     const Rational &rcRational);
  private:
    ...
};
```

Overload Insertion Operator <<

```
std::ostream& operator<< (std::ostream &rcOutput,
                          const Rational &rcRational)
{
  rcOutput << rcRational.mNumerator << '/'
           << rcRational.mDenominator;

  return rcOutput;
}
```

Overload Problem for Rational

Assume Rational cR1 (1, 2), cR2 (2,3), cR3;

1. Overload the insertion operator <<
   std::cout << cR1; // prints **1/2**
2. Overload the extraction operator >>
   std::cin >> cR1; // can enter **1/2** from keyboard
3. Overload the multiplication operator *
   cR3 = cR1 * cR2; // cR3 is now **2/6**
4. Write a driver to test all functions including overloaded operators