Operator Overloading

Section 11.6

Operator Overloading

- Earlier in the semester we created a class for rational numbers
- An example of how a client would use that class is:

```
Rational cRat1(3, 4);
Rational cRat2(2, 5);
Rational cRat3, cRat4;
cRat3 = cRat1.multiplication(cRat2);
cRat4 = cRat1.addition(cRat2);
```

- It would be much easier if we could instead write

```
cRat3 = cRat1 * cRat2;
cRat4 = cRat1 + cRat2;
```


Operator Overloading

- We defined a print function to output the contents of a set

```
cRat1.printRational();
```

- Wouldn't it be more efficient and more consistent with C++ if we could write

```
cout << cRat1;
```

The How of Operator Overloading

- Write a function definition for the operator, but the function name becomes **`operator`** followed by the symbol
  - **`operator<<`**
  - **`operator+`**
  - **`operator==`**

Operator Overloading

- Operator overloading can be achieved in one of two ways
  - A member function of the class
  - A friend function of the class
- Using operator overloading through member functions has the restriction that the object of the class must always be to the left of the operator
  - Not useful for the insertion operator **<<**

`friend` Functions

- 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

`friend` Functions

- The function prototype is placed in the class, preceded by the keyword **`friend`**
- The function is able to directly access the private data members

`friend` Functions

- The **`friend`** function could be a member function in another class
- A class could also be made a friend of an existing class
  - In this case, every member function of the friend class will have access to this class's private data

`operator<<`

- **`<<`** must be overloaded using friend functions
- The return value of **`operator<<`** is an **`ostream&`**
- The arguments will be the output stream and an object of the class

Example

```
class PhoneNumber
{
  friend ostream &operator<<(ostream&,
                             const PhoneNumber &);
  friend istream &operator>>(istream&, PhoneNumber &);
private:
   char areaCode[4];  // 3-digit area code and null
   char exchange[4];  // 3-digit exchange and null
   char line[5];      // 4-digit line and null
};
```


Definition

```
ostream &operator<<(ostream &output, const
  PhoneNumber &num)
{
   output << "(" << num.areaCode << ") "
          << num.exchange << "-" << num.line;
   return output;
}
```


Driver

```
int main()
{
   PhoneNumber phone;
   cout << "The phone number is: ";
   cout << phone << endl;
   return 0;
}
```

Your Turn

```
class Rational
{
public:
   Rational(int = 0, int = 1);
   Rational addition(const Rational &);
   Rational subtraction(const Rational &);
   Rational multiplication(const Rational &);
   Rational division(const Rational &);
   void printRational ();
private:
   int numerator;
   int denominator;
   void reduction();
};
```

- Replace the **`printRational()`** function with **`operator<<`**

Overloading Binary Operators

- Examples of binary operators that can be overloaded are +, -, *, and /
- Unlike the insertion and extraction operators that are overloaded as friend functions, the binary operators are overloaded as regular member functions of the class

Example

- Let us add functionality to the Rational class to support the following:

```
Rational cRat1(3, 4);
Rational cRat2(2, 9);
Rational cRat3;
cRat3 = cRat1 + cRat2;
```

Member Function Prototype

- In the class interface, let us add the function prototype for the overloaded operator

```
RationalNumber operator+(const
                         RationalNumber &);
```


Member Function Definition

```
RationalNumber operator+(const
                         RationalNumber & r)
{
  RationalNumber add;
  add.numerator = numerator * r.denominator +
                  denominator * r.nominator;
  add.denominator = denominator * r.denominator;
  add.reduction();
  return add;
}
```


Your Turn

- Overload the multiplication operator in the rational class