11.5 OVERLOADING THE ARITHMETIC ASSIGNMENT OPERATORS

C++ allows you to combine arithmetic operations with the assignment operator; for example, using `x *= y` in place of `x = x * y`. These combination operators can all be overloaded for use in your own classes.

EXAMPLE 11.6 The `Ratio` Class with an Overloaded `*=` Operator

```
class Ratio
{ public:
    Ratio(int =0, int =1);
    Ratio& operator=(const Ratio&);
    Ratio& operator*=(const Ratio&);
    // other declarations go here
  private:
    int num, den;
    // other declarations go here
};
Ratio& Ratio::operator*=(const Ratio& r)
{ num = num*r.num;
  den = den*r.den;
  return *this;
}
```

The operator `operator*=` has the same syntax and nearly the same implementation as the basic assignment operator `operator=`. By returning `*this`, the operator can be chained, like this:

```
x *= y *= z;
```

It is also important to ensure that overloaded operators perform consistently with each other. For example, the following two lines should have the same effect, even though they call different operators:

```
x = x*y;
x *= y
```

11.6 OVERLOADING THE RELATIONAL OPERATORS

The six relational operators `<`, `>`, `<=`, `>=`, `==`, and `!=` can be overloaded the same way that the arithmetic operators are overloaded: as `friend` functions.

EXAMPLE 11.7 Overloading the Equality Operator `==` in the `Ratio` Class

Like other `friend` functions, the equality operator is declared above the `public` section of the class:

```
class Ratio
{   friend bool operator==(const Ratio&, const Ratio&);
    friend Ratio operator*(const Ratio&, const Ratio&);
    // other declarations go here
  public:
    Ratio(int =0, int =1);
    Ratio(const Ratio&);
    Ratio& operator=(const Ratio&);
    // other declarations go here
    private:
      int num, den;
      // other declarations go here
  };
  bool operator==(const Ratio& x, const Ratio& y)
  { return (x.num * y.den == y.num * x.den);
  }
```

The test for equality of two fractions *a*/*b* and *c*/*d* is equivalent to the test *a***d* == *b***c*. So we end up using the equality operator for `int`s to define the equality operator for `Ratio`s.

Note that the relational operators return an `int` type, representing either "true" (1) or "false" (0).

11.7 OVERLOADING THE STREAM OPERATORS

C++ allows you to overload the stream insertion operator `>>` for customizing input and the stream deletion operator `<<` for customizing output. Like the arithmetic and relational operators, these should also be declared as `friend` functions.

For a class `T` with data member `d`, the syntax for the output operator is

```
  friend ostream& operator<<(ostream& ostr, const T& t)
  { return ostr << t.d; }
```

Here, `ostream` is a standard class defined (indirectly) in the `iostream.h` header file. Note that all the parameters and the return value are passed by reference.

This function can then be called using the same syntax that we used for fundamental types:

```
  cout << "x = " << x << ", y = " << y << endl;
```

Here is an example of how custom output can be written:

EXAMPLE 11.8 Overloading the Output Operator `<<` for the `Ratio` Class

```
  class Ratio
  {   friend ostream& operator<<(ostream&, const Ratio&);
    public:
      Ratio(int n=0, int d=1) : num(n), den(d) { }
      // other declarations go here
    private:
      int num, den;
      // other declarations go here
  };
  int main()
  { Ratio x(22,7), y(-3,8);
    cout << "x = " << x << ", y = " << y << endl;
  }
  ostream& operator<<(ostream& ostr, const Ratio& r)
  { return ostr << r.num << '/' << r.den;
  }
```

```
  x = 22/7, y = -3/8
```

When the second line of `main()` executes, the expression `cout << "x = "` executes first. This calls the standard output operator `<<`, passing the standard output stream `cout` and the string `"x = "` to it. As usual, this inserts the string into the output stream and then returns a reference to `cout`. This return value is then passed with the object `x` to the overloaded `<<` operator. This call to `operator<<` executes with `cout` in place of `ostr` and with `x` in place of `r`. The result is the execution of the line