```
  const Ratio P(22,7);
  const double PI = double(P);
  cout << "P = " << P << ", PI = " << PI << endl;
}

Ratio::operator double() const
{ return double(num)/den;
}
```

```
x = -5/8, double(x) = -0.625
P = 22/7, PI = 3.14286
```

First we use the conversion operator `double()` to convert the `Ratio` object `x` into the `double` -0.625. Then we use it again to convert the constant `Ratio` object `p` into the constant `double pi`.

11.9 OVERLOADING THE INCREMENT AND DECREMENT OPERATORS

The increment operator `++` and the decrement operator `--` each have two forms: prefix and postfix. Each of these four forms can be overloaded. We'll examine the overloading of the increment operator here. Overloading the decrement operator works the same way.

When applied to integer types, the pre-increment operator simply adds 1 to the value of the object being incremented. This is a unary operator: its single argument is the object being incremented. The syntax for overloading it for a class named `T` is simply

```
T operator++();
```

So for our `Ratio` class, it is declared as

```
Ratio operator++();
```

EXAMPLE 11.11 Adding a Pre-Increment Operator to the `Ratio` Class

This example adds an overloaded pre-increment operator `++` to our `Ratio` class. Although we can make this function do whatever we want, it should be consistent with the action that the standard pre-increment operator performs on integer types. That adds 1 to the current value of the object before that value is used in the expression. This is equivalent to adding its denominator to its numerator:

$$\frac{22}{7} + 1 = \frac{22 + 7}{7} = \frac{29}{7}$$

So, we simply add `den` to `num` and then return `*this`, which is the object itself:

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