OVERLOADING operator<< AND operator>>

Common practices and patterns

Delroy A. Brinkerhoff

CATEGORIZING OPERATOR ELEMENTS

- operator<< and operator>> consist of elements following a ridged pattern
- Organizing the elements into four categories makes it easier to present, learn, and use them:
- Categories
  - Unchanging elements
  - The name of the defining or befriending class
  - Programmer-chosen names
  - Function-specific tasks

UNCHANGING FUNCTION ELEMENTS

```
friend ostream& operator<<(ostream& out, fraction& f)
{
    // format and print f's members
    return out;
}
```

```
friend istream& operator>>(istream& in, fraction& f)
{
    // read data into f's members
    return in;
}
```

- Always friend functions
- Always have a stream reference return type
- The first parameter is always a stream reference
- Always ends by returning the first parameter

BEFRIENDING CLASS NAME

```
friend ostream& operator<<(ostream& out, fraction& f)
{
    // format and print f's members
    return out;
}
```

```
friend istream& operator>>(istream& in, fraction& f)
{
    // read data into f's members
    return in;
}
```

- The pattern requires a second reference parameter
- Names the class defining or befriending the function

PROGRAMMER-CHOSEN PARAMETER NAMES

```
friend ostream& operator<<(ostream& out, fraction& f)
{
    // format and print f's members
    return out;
}
```

```
friend istream& operator>>(istream& in, fraction& f)
{
    // read data into f's members
    return in;
}
```

- Programmers choose appropriate parameter names

FUNCTION-SPECIFIC TASKS

```
friend ostream& operator<<(ostream& out, fraction& f)
{
    // format and print f's members
    return out;
}
```

```
friend istream& operator>>(istream& in, fraction& f)
{
    // read data into f's members
    return in;
}
```

- Functions are designed for specific objects or class instances
- A function's exact operation depends on the befriending class's members

SIMPLIFIED fraction CLASS

```
class fraction
{
    private:
        int numerator;
        int denominator;
    public:
        friend ostream& operator<<(ostream& out, fraction& f);
        friend istream& operator>>(istream& in, fraction& f);
    private:
        void reduce();
};
```

INSERTER

```
ostream& operator<<(ostream& out, fraction& f)
{
    out << f.denominator << "/" << f.numerator;

    return out;
}
```

EXTRACTOR

```
istream& operator>>(istream& in, fraction& f)
{
    cout << "Please enter the numerator: ";
    in >> f.numerator;
    cout << "Please enter the denominator: ";
    in >> f.denominator;
    f.reduce();

    return in;
}
```

MAPPING ARGUMENTS TO PARAMETERS

CLIENT CODE

```
fraction left;
fraction right;

cin >> left;
cin >> right;
fraction result = left + right;
cout << result << endl;
```

MAPPING