Arithmetic and Relational Operators

Ordinarily, we define the arithmetic and relational operators as nonmember funcitons in order to allow conversions for either the left- or right-hand operand. These operators shouldn't need to change the state of either operand, so the parameters are ordinarily references to `const`.

Classes that define an arithmetic operator generally define the corresponding compound assignment operator as well.

```
// It would be better to define as member funtion
Foo operator+(const Foo &,const Foo &);
Foo operator-(const Foo &,const Foo &);
Foo operator*(const Foo &,const Foo &);
Foo operator/(const Foo &,const Foo &);

// compound operators
Foo operator+=(const Foo &,const Foo &);

Foo operator+(const Foo &,const Bar &);
Foo operator+=(const Foo &,const Bar &);
//...
```

Equality Operators

It will be define as nonmember function as well.

```
bool operator==(const Foo &,const Foo &);
bool operator!=(const Foo &,const Foo &);
```

- ordinarily, the equality operator should be transitive, meaning that if `a==b` and `b==c` are both true, the `a==c` should also be true.
- One of the equality or inequality operators should delegate the work to the other. That is, one of these operators should do the real work to compare objects. The other should call the one that does the real work.

```
bool operator==(const Foo &f,const Foo &f_){
    return f.data==f_.data;
}

bool operator!=(const Foo &f,const Foo &f_){
    return !(f==f_);
}
```

Relational Operators

Classes for which the equality operators is defined also often (but not always) have relational operators.

Ordinarily, the relational operators should

1. define an ordering relation that is consistent with the requirements for use as a key to an associative container;
2. define a relation that is consistent with `==` if the class has both operators. In particular, if two objects are `!=`, then one object should be `<` the other.