Increment and Decrement Operators

The increment (`++`) and decrement (`--`) operators let the class move between the elements of a sequence. There is no language requirement that these operatos be members of the class. However, because these operators change the object on which they operate, our preference is to make them members.

For the built-in types, there are both **prefix** and **postfix** versions of the increment and decrement operators. Not surprisingly, we can define both the prefix and postfix instance of these operations for our own classes as well.

Defining Prefix Increment/Decrement Operators

An example:

```
class Foo{
    public:
        // prefix version
        Foo &operator++();
        Foo &operator--();
        // return a reference to the incremented or decremented object.
}
```

The increment and decrement operators work similarly to each other-they call `check` to verify that the `Foo` is still valid. If `check` doesn't throw an exception, these operators return a reference to this object.

In the case of increment, we pass the current value of `curr` to `check`. So long as that value is less that the size of the underlying `vector,check` will return. If `curr` is already at the end of the `vector`, `check` will throw:

```
// prefix: return a reference to the incremented/decremented object
Foo & Foo::operator++(){
    // if curr already points past the end of the container, can't increment it
    check(curr);
    ++curr; // advance the current state
    return *this;
}

Foo & Foo::operator--(){
    --curr; // move the current state back one element
    check(curr); // if curr is zero, decrementing it will yield an invalid
subscript
    return *this;
}
```

Defferentiating Prefix and Postfix Operators

There is one problem with defining both the prefix and postfix operators: Normal overloading connot distinguish between these operators. The prefix and postfix versions use the same symbol, meaning that the overloaded

versions of these operators have the same name. They also have the same number and type of operands.

To solve this problem, the postfix version take an extra (unused) parameter of type `int`. When we use a postfix operator, the compiler supplies `0` as the argument for this parameter. Although the postfix function can use this extra parameter, it usually should not. That parameter is not needed for the work normally performed by a postfix operator. *Its sole purpose is to distinguish a postfix function from the prefix version.*

```
class Foo{
    public:
        // postfix version
        Foo operator++(int);
        Foo operator--(int);
};
```

> To be consistent with the built-in operators, the postfix operators should return the old value (unincremented or undecrenmented). That value is returned as a value, not a reference.

Thus, the postfix versions have to remember the current state of the object before incrementing the object:

```
Foo Foo::operator++(int){
    // no check here; the call to prefix increment will do the check

    Foo ret=*this; // save the currenet value;

    ++*this;// check by calling the prefix increment operator

    return ret; // return the saved state
}

Foo Foo::operator--(int){
    Foo ret=*this;
    --*this;
    return ret;
}
```

The postfix functions return the stored copy in `ret`. Thus, after the return, the object itself has been advanced, but the value returned reflects the original, unincremented value.

> Note: the `int` parameter is not used, so we do not give it a name.

Calling the Postfix Operators Explicitly

```
Foo f;
f.operator++(0);// call postfix operator ++
f.operator++();// call prefix operator ++
```

The value passed usually is ignored but is necessary in order to tell the compiler to use the postfix version.