# Assignment Operators

In addtion to the copy-and move-assignment operators that assign one object of the class type to another object of the same type, *a class can define additional assignment operators that allow other types as the right-hand operand.*

```
vector<string> v;
v={"a","an","the"}; // the third assignment operator in vector class
```

Adding this operator to `StrVec` class is shown as follows.

```
class StrVec{
    public:
        StrVec &operator=(std::initializer_list<std::string>);
        // other members
};
StrVec & StrVec::operator=(std::initializer_list<std::string> il){
    // alloc_n_copy allocates space and copies elements from the given range
    auto data=alloc_n_copy(il.begin(),il.end());
    free(); // destroy the elements in this object and free the space

    elements=data.first;
    first_free=cap=data.second;
    return *this;
}
```

> Note: assignment operators can be overloaded. Assignment operators, regradless of parameter type, must be defined as member functions.

**Compound-Assignment Operators**

Compound assignment operators are not required to be members. However, we prefer to define all assignments, including compound assignments, in the class. For consistency with the built-in compound assignment, these operators should return a reference to their left-hand operand.

```
Foo & Foo::operator+=(const Foo &rhs){// the first parameter is implicitly this
object
    this->val+=rhs.val;
    // ...
    return *this;
}
```