C++ Notes: OOP:Overloading Assignment

When to define assignment (and a copy constructor and destructor)

If your object has a pointer to **dynamically allocated memory**, eg allocated in the constructor, you will want to make a *deep copy* of the object. Deep copies require overloading assignment, as well as defining a copy constructor and a destructor).

Pattern

The following steps are a typical for the assignment operator.

1. **No self-assignment**. Test to make sure you aren't assigning an object to itself, eg `x = x`. Of course no one would write that statement, but in can happen when one side of the assignment is not so obviously the same object. Self assignment fails because the memory associated with the current value of the left-hand-side is deallocated before the assignment, which would invalidate using it from the right-hand-side.
2. **Delete** the associated memory. Same as copy constructor.
3. **Copy** all the members. Same as copy constructor.
4. **Return** `*this`. This is necessary to allow multiple assignment, eg `x = y = z;`

Example

```
//--- file Person.h
. . .
class Person {
    private:
        char* _name;
        int   _id;
    public:
        Person& Person::operator=(const Person& p);
    . . .
}

//--- file Person.cpp
. . .
//=================================================== operator=
Person& Person::operator=(const Person& p) {
    if (this != &p) {  // make sure not same object
        delete [] _name;                     // Delete old name's memory.
        _name = new char[strlen(p._name)+1]; // Get new space
        strcpy(_name, p._name);              // Copy new name
        _id = p._id;                         // Copy id
    }
    return *this;    // Return ref for multiple assignment
}//end operator=
```

More

See Overloading Derived Class Assignment for the secret to getting a parent's members assigned.