Image: iStockPhoto #6648802.
Per 10.3/2, what I call “adornments” are officially called \textit{cv-qualifications} and \textit{ref-qualifiers}.
Overriding Functions

Overrider may omit virtual:

class Base {
  public:
    virtual void f() const;
};
class Derived1: public Base {
  public:
    virtual void f() const;  // overrides Base::f
};
class Derived2: public Base {
  public:
    void f() const;  // also overrides Base::f
};
None of VC10, VC11, and gcc 4.7 issue a warning for this code.
None of VC10, VC11, and gcc 4.7 issue a warning for this code.
None of VC10, VC11, and gcc 4.7 issue a warning for this code.
override-declared functions must override inherited versions:

```cpp
class Base {
    public:
        virtual void f() const;
};
class Derived1: public Base {
    public:
        void f() const override;  // fine
};
class Derived2: public Base {
    public:
        virtual void f() override;  // error! not an override
        // (missing const)
};
```

override goes after cv- and ref qualifications.

Declaring an override function virtual is redundant. override functions are always virtual.
override

Prevents all earlier errors from compiling:

- Attempt to override non-virtual function:
  ```
  class Base {
  public:
    void f() const;
  }
  
  class Derived: public Base {
  public:
    void f() const override;  // error!
  }
  ```
override

- Mismatched parameter types in base and derived classes:
  ```cpp
class Base {
public:
  virtual void f(long) const;
};
class Derived: public Base {
public:
  virtual void f(int) const override;     // error!
};
```
**override**

- Mismatched `const/volatile` declarations in base and derived classes:

```cpp
class Base {
public:
    virtual void f(int) const;
};
class Derived: public Base {
public:
    virtual void f(int) override;  // error!
};
```
Other benefits:

- Helps identify virtuals in derived classes.

```cpp
class Derived: public Base {
public:
    void f1();    // virtual?
    void f2() override;  // virtual!
...
};
```

- Identifies affected derived functions if base signature changes.
  - Reduces maintenance-induced errors.
override

Covariant return types remain legal:

class Base {
  public:
    virtual Base& me();
};
class Derived: public Base {
  public:
    virtual Derived& me() override;       // fine
};
The only reason I can think of for having code like this in your system is that it’s legacy, and you don’t want to change it.
Fun with "override"

Legal code you don’t want to write:

```cpp
class override {};

class Base {
public:
    virtual ::override override(::override); // override takes
    // and returns an
    // ::override
};

class Derived: public Base {
public:
    ::override override(::override) override; // an override
    // of above :-) 
};
```

This code compiles cleanly with both gcc 4.7 and VC11.
Guideline

Declare overriding functions override.