Cleanup of *auto_ptr* copy semantics. X3J16/96-0012 WG21/N0830

Greg Colvin. Information Mananagement Research.

Our working paper's *auto_ptr* (20.4.5) specifies a non-*const* copy constructor and assignment operator. This specification makes functions that return *auto_ptr* less useful than I intended, due to a language restriction on modifying temporaries (8.5.3). I know of no small change to the language that allows *auto_ptr* work as intended and still disallows the silliness that the current restriction prohibits, so a small change to *auto_ptr* is in order.

The currrent *auto_ptr* semantics transfers the ownership and possesion of the held pointer in a "destructive copy": after the copy the source holds a null pointer. The smallest change I know of that preserves a semantics of strict ownership is to separate the concepts of "holding a pointer" and "owning an object", so that more than one *auto_ptr* can hold a pointer to an object, but only one *auto_ptr* can own the object. Copying transfers ownership but does not invalidate the source.

Note that only the held pointer is directly accessible: which *auto*_ptr owns an object is observable at run-time only via the side effects of destructor calls (though it should be known to the programmer by design). Thus providing *const* arguments to the copy constructor and assignment operator and making *release()* a *const* function seem to me a reasonable case of "logical constness". Note also that with this change the idiom *p.reset(q.release())* cannot safely transfer ownership: if *q* is not the owner *p* will nonetheless become an owner. The safest course is to remove the *reset()* function from the interface, as its functionality is now subsumed by the assignment operator. Except for this deletion, existing code, including the motivating examples from our earlier discussions, continues to work unchanged.

The changed Working Paper text I propose is as follows.

20.4.5 Template class *auto_ptr* [lib.auto.ptr]

Template *auto_ptr* holds onto a pointer to an object obtained via *new* and deletes that object when it is the owner of that object and is itself destroyed (such as when leaving block scope 6.7).

```
namespace std {
  template<class X> class auto_ptr {
  public:
  // 20.4.5.1 construct/copy/destroy:
    explicit auto_ptr(X* p=0);
    template<class Y> auto_ptr(const auto_ptr<Y>&);
    template<class Y> auto_ptr& operator=(const auto_ptr<Y>&);
    ~auto_ptr();
  // 20.4.5.2 members:
    X& operator*() const;
    X* operator->() const;
    X* get() const;
    X* release() const;
  };
}
```

The *auto_ptr* provides a semantics of strict ownership. After initial construction an *auto_ptr* owns the object it holds a pointer to. An object may be safely owned by only one *auto_ptr*, so copying an *auto_ptr* copies the pointer and transfers ownership to the destination.

20.4.5.1 *auto_ptr* constructors [lib.auto.ptr.cons]

```
explicit auto_ptr(X* p = 0);
```
Requires: *p* points to an object of type *X* or a class derived from *X* for which *delete p* is defined and accessible, or else *p* is a null pointer.
Postconditions: **this* holds the pointer *p*. **this* is the owner of the object ***this*.

```
template<class Y> auto_ptr(const auto_ptr<Y>& a);
```
Requires: *Y* is type *X* or a class derived from *X* for which *delete(Y*)* is defined and accessible.
Effects: Calls *a.release()*.
Postconditions: **this* holds the pointer returned from *a.release()*. **this* is the owner of the object ***this*.

```
template<class Y>auto_ptr<X>& operator=(const auto_ptr<Y>& a);
```
Requires: *Y* is type *X* or a class derived from X for which *delete(Y*)* is defined and accessible.
Effects: If **this* is the owner of ***this* and *this != &a* then *delete get()*. Calls *a.release()*.
Returns: **this*
Postconditions: **this* holds the pointer returned from *a.release()*. **this* is the owner of the object ***this*.

```
~auto_ptr();
```
Effects: If **this* is the owner of ***this* then *delete get()*.

20.4.5.2 *auto_ptr* members [lib.auto.ptr.members]

```
X& operator*() const;
```
Requires: *get() != 0*
Returns: **get()*

```
X* operator->() const;
```
Returns: *get()*

```
X* get() const;
```
Returns: The pointer **this* holds.

```
X* release() const;
```
Returns: *get()*
Postcondition: **this* is not the owner of the object ***this*.