```
/* The following code example is taken from the book
 * "The C++ Standard Library - A Tutorial and Reference"
 * by Nicolai M. Josuttis, Addison-Wesley, 1999
 *
 * (C) Copyright Nicolai M. Josuttis 1999.
 * Permission to copy, use, modify, sell and distribute this software
 * is granted provided this copyright notice appears in all copies.
 * This software is provided "as is" without express or implied
 * warranty, and with no claim as to its suitability for any purpose.
 */
/* class auto_ptr
 * - improved standard conforming implementation
 */
namespace std {
    // auxiliary type to enable copies and assignments (now global)
    template<class Y>
    struct auto_ptr_ref {
        Y* yp;
        auto_ptr_ref (Y* rhs)
         : yp(rhs) {
        }
    };

    template<class T>
    class auto_ptr {
      private:
        T* ap;    // refers to the actual owned object (if any)
      public:
        typedef T element_type;

        // constructor
        explicit auto_ptr (T* ptr = 0) throw()
         : ap(ptr) {
        }

        // copy constructors (with implicit conversion)
        // - note: nonconstant parameter
        auto_ptr (auto_ptr& rhs) throw()
         : ap(rhs.release()) {
        }
        template<class Y>
        auto_ptr (auto_ptr<Y>& rhs) throw()
         : ap(rhs.release()) {
        }

        // assignments (with implicit conversion)
        // - note: nonconstant parameter
        auto_ptr& operator= (auto_ptr& rhs) throw() {
            reset(rhs.release());
            return *this;
        }
        template<class Y>
        auto_ptr& operator= (auto_ptr<Y>& rhs) throw() {
            reset(rhs.release());
            return *this;
        }
        // destructor
        ~auto_ptr() throw() {
            delete ap;
        }

        // value access
        T* get() const throw() {
            return ap;
        }
        T& operator*() const throw() {
            return *ap;
        }
        T* operator->() const throw() {
            return ap;
        }

        // release ownership
        T* release() throw() {
            T* tmp(ap);
            ap = 0;
            return tmp;
        }

        // reset value
        void reset (T* ptr=0) throw() {
            if (ap != ptr) {
                delete ap;
                ap = ptr;
            }
        }

        /* special conversions with auxiliary type to enable copies and
assignments
         */
        auto_ptr(auto_ptr_ref<T> rhs) throw()
         : ap(rhs.yp) {
        }
        auto_ptr& operator= (auto_ptr_ref<T> rhs) throw() {  // new
             reset(rhs.yp);
             return *this;
        }
        template<class Y>
        operator auto_ptr_ref<Y>() throw() {
            return auto_ptr_ref<Y>(release());
        }
        template<class Y>
        operator auto_ptr<Y>() throw() {
            return auto_ptr<Y>(release());
        }
    };
}
```