```
/* 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.
 */
#ifndef COUNTED_PTR_HPP
#define COUNTED_PTR_HPP

/* class for counted reference semantics
 * - deletes the object to which it refers when the last CountedPtr
 *   that refers to it is destroyed
 */
template <class T>
class CountedPtr {
  private:
    T* ptr;        // pointer to the value
    long* count;   // shared number of owners

  public:
    // initialize pointer with existing pointer
    // - requires that the pointer p is a return value of new
    explicit CountedPtr (T* p=0)
     : ptr(p), count(new long(1)) {
    }

    // copy pointer (one more owner)
    CountedPtr (const CountedPtr<T>& p) throw()
     : ptr(p.ptr), count(p.count) {
        ++*count;
    }

    // destructor (delete value if this was the last owner)
    ~CountedPtr () throw() {
        dispose();
    }

    // assignment (unshare old and share new value)
    CountedPtr<T>& operator= (const CountedPtr<T>& p) throw() {
        if (this != &p) {
            dispose();
            ptr = p.ptr;
            count = p.count;
            ++*count;
        }
        return *this;
    }

    // access the value to which the pointer refers
    T& operator*() const throw() {
        return *ptr;
    }
```

```
    T* operator->() const throw() {
        return ptr;
    }

  private:
    void dispose() {
        if (--*count == 0) {
            delete count;
            delete ptr;
        }
    }

};

#endif /*COUNTED_PTR_HPP*/
```