G++ 2.91.57，cygnus\cygwin-b20\include\g++**memory** 完整列表

```
/*
 * Copyright (c) 1997
 * Silicon Graphics Computer Systems, Inc.
 *
 * Permission to use, copy, modify, distribute and sell this software
 * and its documentation for any purpose is hereby granted without fee,
 * provided that the above copyright notice appear in all copies and
 * that both that copyright notice and this permission notice appear
 * in supporting documentation.  Silicon Graphics makes no
 * representations about the suitability of this software for any
 * purpose.  It is provided "as is" without express or implied warranty.
 *
 */

#ifndef __SGI_STL_MEMORY
#define __SGI_STL_MEMORY

#include <stl_algobase.h>
#include <stl_alloc.h>
#include <stl_construct.h>
#include <stl_tempbuf.h>
#include <stl_uninitialized.h>
#include <stl_raw_storage_iter.h>

// Note: auto_ptr is commented out in this release because the details
//  of the interface are still being discussed by the C++ standardization
//  committee.  It will be included once the iterface is finalized.

#if 0
#if defined(_MUTABLE_IS_KEYWORD) && defined(_EXPLICIT_IS_KEYWORD) && \
    defined(__STL_MEMBER_TEMPLATES)

__STL_BEGIN_NAMESPACE

template <class X> class auto_ptr {
private:
  X* ptr;
  mutable bool owns;
public:
  typedef X element_type;
  explicit auto_ptr(X* p = 0) __STL_NOTHROW : ptr(p), owns(p) {}
  auto_ptr(const auto_ptr& a) __STL_NOTHROW : ptr(a.ptr), owns(a.owns) {
    a.owns = 0;
  }
  template <class T> auto_ptr(const auto_ptr<T>& a) __STL_NOTHROW
    : ptr(a.ptr), owns(a.owns) {
      a.owns = 0;
```

```
  }

  auto_ptr& operator=(const auto_ptr& a) __STL_NOTHROW {
    if (&a != this) {
      if (owns)
        delete ptr;
      owns = a.owns;
      ptr = a.ptr;
      a.owns = 0;
    }
  }
  template <class T> auto_ptr& operator=(const auto_ptr<T>& a) __STL_NOTHROW {
    if (&a != this) {
      if (owns)
        delete ptr;
      owns = a.owns;
      ptr = a.ptr;
      a.owns = 0;
    }
  }
  ~auto_ptr() {
    if (owns)
      delete ptr;
  }

  X& operator*() const __STL_NOTHROW { return *ptr; }
  X* operator->() const __STL_NOTHROW { return ptr; }
  X* get() const __STL_NOTHROW { return ptr; }
  X* release const __STL_NOTHROW { owns = false; return ptr }
};

__STL_END_NAMESPACE
#endif /* mutable && explicit && member templates */
#endif /* 0 */


#endif /* __SGI_STL_MEMORY */


// Local Variables:
// mode:C++
// End:
```