G++ 2.91.57，cygnus\cygwin-b20\include\g++**defalloc.h** 完整列表

```
/*
 *
 * Copyright (c) 1994
 * Hewlett-Packard Company
 *
 * 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.  Hewlett-Packard Company makes no
 * representations about the suitability of this software for any
 * purpose.  It is provided "as is" without express or implied warranty.
 *
 */

// Inclusion of this file is DEPRECATED.  This is the original HP
// default allocator.  It is provided only for backward compatibility.
//
// DO NOT USE THIS FILE unless you have an old container implementation
// that requires an allocator with the HP-style interface.  SGI STL
// uses a different allocator interface.  SGI-style allocators are not
// parametrized with respect to the object type; they traffic in void *
// pointers.  This file is not included by any other SGI STL header.

#ifndef DEFALLOC_H
#define DEFALLOC_H

#include <new.h>
#include <stddef.h>
#include <stdlib.h>
#include <limits.h>
#include <iostream.h>
#include <algobase.h>


template <class T>
inline T* allocate(ptrdiff_t size, T*) {
    set_new_handler(0);
    T* tmp = (T*)(::operator new((size_t)(size * sizeof(T))));
    if (tmp == 0) {
         cerr << "out of memory" << endl;
         exit(1);
    }
    return tmp;
}


template <class T>
```

```
inline void deallocate(T* buffer) {
    ::operator delete(buffer);
}

template <class T>
class allocator {
public:
    typedef T value_type;
    typedef T* pointer;
    typedef const T* const_pointer;
    typedef T& reference;
    typedef const T& const_reference;
    typedef size_t size_type;
    typedef ptrdiff_t difference_type;

    pointer allocate(size_type n) {
        return ::allocate((difference_type)n, (pointer)0);
    }
    void deallocate(pointer p) { ::deallocate(p); }
    pointer address(reference x) { return (pointer)&x; }
    const_pointer const_address(const_reference x) {
        return (const_pointer)&x;
    }
    size_type init_page_size() {
        return max(size_type(1), size_type(4096/sizeof(T)));
    }
    size_type max_size() const {
        return max(size_type(1), size_type(UINT_MAX/sizeof(T)));
    }
};

class allocator<void> {
public:
    typedef void* pointer;
};


#endif
```