G++ 2.91.57，cygnus\cygwin-b20\include\g++**stl_tempbuf.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.
 *
 *
 * Copyright (c) 1996,1997
 * 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.
 */

/* NOTE: This is an internal header file, included by other STL headers.
 *   You should not attempt to use it directly.
 */

#ifndef __SGI_STL_INTERNAL_TEMPBUF_H
#define __SGI_STL_INTERNAL_TEMPBUF_H


__STL_BEGIN_NAMESPACE

template <class T>
pair<T*, ptrdiff_t> get_temporary_buffer(ptrdiff_t len, T*) {
  if (len > ptrdiff_t(INT_MAX / sizeof(T)))
    len = INT_MAX / sizeof(T);

  while (len > 0) {
    T* tmp = (T*) malloc((size_t)len * sizeof(T));
    if (tmp != 0)
      return pair<T*, ptrdiff_t>(tmp, len);
    len /= 2;
  }
```

```
  return pair<T*, ptrdiff_t>((T*)0, 0);
}

template <class T>
void return_temporary_buffer(T* p) {
  free(p);
}

template <class ForwardIterator,
          class T
#ifdef __STL_CLASS_PARTIAL_SPECIALIZATION
                = iterator_traits<ForwardIterator>::value_type
#endif /* __STL_CLASS_PARTIAL_SPECIALIZATION */
         >
class temporary_buffer {
private:
  ptrdiff_t original_len;
  ptrdiff_t len;
  T* buffer;

  void allocate_buffer() {
    original_len = len;
    buffer = 0;

    if (len > (ptrdiff_t)(INT_MAX / sizeof(T)))
      len = INT_MAX / sizeof(T);

    while (len > 0) {
      buffer = (T*) malloc(len * sizeof(T));
      if (buffer)
        break;
      len /= 2;
    }
  }

  void initialize_buffer(const T&, __true_type) {}
  void initialize_buffer(const T& val, __false_type) {
    uninitialized_fill_n(buffer, len, val);
  }

public:
  ptrdiff_t size() const { return len; }
  ptrdiff_t requested_size() const { return original_len; }
  T* begin() { return buffer; }
  T* end() { return buffer + len; }

  temporary_buffer(ForwardIterator first, ForwardIterator last) {
    __STL_TRY {
      len = 0;
```

```
      distance(first, last, len);
      allocate_buffer();
      if (len > 0)
        initialize_buffer(*first,
                          typename __type_traits<T>::has_trivial_default_constructor());
    }
    __STL_UNWIND(free(buffer); buffer = 0; len = 0);
  }

  ~temporary_buffer() {
    destroy(buffer, buffer + len);
    free(buffer);
  }

private:
  temporary_buffer(const temporary_buffer&) {}
  void operator=(const temporary_buffer&) {}
};

__STL_END_NAMESPACE

#endif /* __SGI_STL_INTERNAL_TEMPBUF_H */

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