

G++ 2.91.57, cygnus\cygwin-b20\include\g++\stl_uninitialized.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_UNINITIALIZED_H
#define __SGI_STL_INTERNAL_UNINITIALIZED_H

__STL_BEGIN_NAMESPACE

// 以下是 __uninitialized_copy 的實作

// 如果 copy construction 等同於assignment, 而且
// destructor 是 trivial, 以下就有效。
// 如果是POD 型別, 執行流程就會轉進到以下函式。這是藉由function template
// 的引數推導機制而得。
template <class InputIterator, class ForwardIterator>
inline ForwardIterator
__uninitialized_copy_aux(InputIterator first, InputIterator last,
                        ForwardIterator result,
                        __true_type) {
    return copy(first, last, result); // 呼叫 STL 演算法 copy()
}
```

```

// 如果是non-POD型別，執行流程就會轉進到以下函式。這是藉由function template
// 的引數推導機制而得。
template <class InputIterator, class ForwardIterator>
ForwardIterator
__uninitialized_copy_aux(InputIterator first, InputIterator last,
                        ForwardIterator result,
                        __false_type) {
    ForwardIterator cur = result;
    __STL_TRY {
        for ( ; first != last; ++first, ++cur)
            construct(&*cur, *first); // 必須一個一個元素地建構，無法批量進行
        return cur;
    }
    __STL_UNWIND(destroy(result, cur));
}

template <class InputIterator, class ForwardIterator, class T>
inline ForwardIterator
__uninitialized_copy(InputIterator first, InputIterator last,
                    ForwardIterator result, T*) {
    typedef typename __type_traits<T>::is_POD_type is_POD;
    return __uninitialized_copy_aux(first, last, result, is_POD());
    // 以上，企圖利用 is_POD() 所獲得的結果，讓編譯器做引數推導。
}

template <class InputIterator, class ForwardIterator>
inline ForwardIterator
uninitialized_copy(InputIterator first, InputIterator last,
                  ForwardIterator result) {
    return __uninitialized_copy(first, last, result, value_type(result));
    // 以上，利用 value_type() 取出 first 的 value type.
}

// 以下是針對 const char* 的特化版本
inline char* uninitialized_copy(const char* first, const char* last,
                               char* result) {
    memmove(result, first, last - first);
    return result + (last - first);
}

// 以下是針對 const wchar_t* 的特化版本
inline wchar_t* uninitialized_copy(const wchar_t* first, const wchar_t* last,
                                   wchar_t* result) {
    memmove(result, first, sizeof(wchar_t) * (last - first));
    return result + (last - first);
}

// 以下是 __uninitialized_copy_n 的實作

```

```

template <class InputIterator, class Size, class ForwardIterator>
pair<InputIterator, ForwardIterator>
__uninitialized_copy_n(InputIterator first, Size count,
                      ForwardIterator result,
                      input_iterator_tag) {
    ForwardIterator cur = result;
    __STL_TRY {
        for ( ; count > 0 ; --count, ++first, ++cur)
            construct(&*cur, *first);
        return pair<InputIterator, ForwardIterator>(first, cur);
    }
    __STL_UNWIND(destroy(result, cur));
}

template <class RandomAccessIterator, class Size, class ForwardIterator>
inline pair<RandomAccessIterator, ForwardIterator>
__uninitialized_copy_n(RandomAccessIterator first, Size count,
                      ForwardIterator result,
                      random_access_iterator_tag) {
    RandomAccessIterator last = first + count;
    return make_pair(last, uninitialized_copy(first, last, result));
}

template <class InputIterator, class Size, class ForwardIterator>
inline pair<InputIterator, ForwardIterator>
uninitialized_copy_n(InputIterator first, Size count,
                    ForwardIterator result) {
    return __uninitialized_copy_n(first, count, result,
                                iterator_category(first));
}

// 以下是 __uninitialized_fill 的實作

// 如果 copy construction 等同於assignment, 而且
// destructor 是 trivial, 以下就有效。
// 如果是POD 型別, 執行流程就會轉進到以下函式。這是藉由function template
// 的引數推導機制而得。
template <class ForwardIterator, class T>
inline void
__uninitialized_fill_aux(ForwardIterator first, ForwardIterator last,
                        const T& x, __true_type)
{
    fill(first, last, x);    // 呼叫 STL 演算法 fill()
}

// 如果是non-POD型別, 執行流程就會轉進到以下函式。這是藉由function template
// 的引數推導機制而得。
template <class ForwardIterator, class T>

```

```

void
__uninitialized_fill_aux(ForwardIterator first, ForwardIterator last,
                        const T& x, __false_type)
{
    ForwardIterator cur = first;
    __STL_TRY {
        for ( ; cur != last; ++cur)
            construct(&*cur, x); // 必須一個一個元素地建構，無法批量進行
    }
    __STL_UNWIND(destroy(first, cur));
}

template <class ForwardIterator, class T, class T1>
inline void __uninitialized_fill(ForwardIterator first, ForwardIterator last,
                                const T& x, T1*) {
    typedef typename __type_traits<T1>::is_POD_type is_POD;
    __uninitialized_fill_aux(first, last, x, is_POD());
}

template <class ForwardIterator, class T>
inline void uninitialized_fill(ForwardIterator first, ForwardIterator last,
                              const T& x) {
    __uninitialized_fill(first, last, x, value_type(first));
}

// 以下是 __uninitialized_fill_n 的實作

// 如果 copy construction 等同於assignment，而且
// destructor 是 trivial，以下就有效。
// 如果是POD 型別，執行流程就會轉進到以下函式。這是藉由function template
// 的引數推導機制而得。
template <class ForwardIterator, class Size, class T>
inline ForwardIterator
__uninitialized_fill_n_aux(ForwardIterator first, Size n,
                          const T& x, __true_type) {
    return fill_n(first, n, x);
}

// 如果是non-POD型別，執行流程就會轉進到以下函式。這是藉由function template
// 的引數推導機制而得。
template <class ForwardIterator, class Size, class T>
ForwardIterator
__uninitialized_fill_n_aux(ForwardIterator first, Size n,
                          const T& x, __false_type) {
    ForwardIterator cur = first;
    __STL_TRY {
        for ( ; n > 0; --n, ++cur)
            construct(&*cur, x);
    }
}

```

```

        return cur;
    }
    __STL_UNWIND(destroy(first, cur));
}

template <class ForwardIterator, class Size, class T, class T1>
inline ForwardIterator __uninitialized_fill_n(ForwardIterator first, Size n,
                                              const T& x, T1*) {
    typedef typename __type_traits<T1>::is_POD_type is_POD;
    return __uninitialized_fill_n_aux(first, n, x, is_POD());
}

template <class ForwardIterator, class Size, class T>
inline ForwardIterator uninitialized_fill_n(ForwardIterator first, Size n,
                                              const T& x) {
    return __uninitialized_fill_n(first, n, x, value_type(first));
}

// Copies [first1, last1) into [result, result + (last1 - first1)), and
// copies [first2, last2) into
// [result, result + (last1 - first1) + (last2 - first2)).

template <class InputIterator1, class InputIterator2, class ForwardIterator>
inline ForwardIterator
__uninitialized_copy_copy(InputIterator1 first1, InputIterator1 last1,
                        InputIterator2 first2, InputIterator2 last2,
                        ForwardIterator result) {
    ForwardIterator mid = uninitialized_copy(first1, last1, result);
    __STL_TRY {
        return uninitialized_copy(first2, last2, mid);
    }
    __STL_UNWIND(destroy(result, mid));
}

// Fills [result, mid) with x, and copies [first, last) into
// [mid, mid + (last - first)).
template <class ForwardIterator, class T, class InputIterator>
inline ForwardIterator
__uninitialized_fill_copy(ForwardIterator result, ForwardIterator mid,
                        const T& x,
                        InputIterator first, InputIterator last) {
    uninitialized_fill(result, mid, x);
    __STL_TRY {
        return uninitialized_copy(first, last, mid);
    }
    __STL_UNWIND(destroy(result, mid));
}

```

```
// Copies [first1, last1) into [first2, first2 + (last1 - first1)), and
// fills [first2 + (last1 - first1), last2) with x.
template <class InputIterator, class ForwardIterator, class T>
inline void
__uninitialized_copy_fill(InputIterator first1, InputIterator last1,
                        ForwardIterator first2, ForwardIterator last2,
                        const T& x) {
    ForwardIterator mid2 = uninitialized_copy(first1, last1, first2);
    __STL_TRY {
        uninitialized_fill(mid2, last2, x);
    }
    __STL_UNWIND(destroy(first2, mid2));
}

__STL_END_NAMESPACE

#endif /* __SGI_STL_INTERNAL_UNINITIALIZED_H */

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