G++ 2.91.57，cygnus\cygwin-b20\include\g++**stl_config.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) 1997
 * Silicon Graphics
 *
 * 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 __STL_CONFIG_H
# define __STL_CONFIG_H

// 本檔所做的事情：
//  (1)  如果編譯器沒有定義 bool, true, false，就定義它們
//  (2)  如果編譯器的標準程式庫未支援drand48() 函式，就定義 __STL_NO_DRAND48
//  (3)  如果編譯器無法處理static members of template classes，就定義
//       __STL_STATIC_TEMPLATE_MEMBER_BUG
//  (4)  如果編譯器未支援關鍵字typename，就將'typename' 定義為一個null macro.
//  (5)  如果編譯器支援partial specialization of class templates，就定義
//       __STL_CLASS_PARTIAL_SPECIALIZATION.
//  (6)  如果編譯器支援partial ordering of function templates（亦稱為
//        partial specialization of function templates），就定義
//        __STL_FUNCTION_TMPL_PARTIAL_ORDER
//  (7)  如果編譯器允許我們在呼叫一個 function template 時可以明白指定其
//       template arguments，就定義 __STL_EXPLICIT_FUNCTION_TMPL_ARGS
//  (8)  如果編譯器支援template members of classes，就定義
//        __STL_MEMBER_TEMPLATES.
//  (9)  如果編譯器不支援關鍵字explicit，就定義'explicit' 為一個null macro.
//  (10) 如果編譯器無法根據前一個template parameters 設定下一個template
//        parameters 的預設值，就定義 __STL_LIMITED_DEFAULT_TEMPLATES
//  (11) 如果編譯器針對non-type template parameters 執行function template
```

```
//      的引數推導（argument deduction）時有問題，就定義
//      __STL_NON_TYPE_TMPL_PARAM_BUG.
//  (12) 如果編譯器無法支援迭代器的 operator->，就定義
//       __SGI_STL_NO_ARROW_OPERATOR
//  (13) 如果編譯器（在你所選擇的模式中）支援exceptions，就定義
//       __STL_USE_EXCEPTIONS
//  (14) 如果我們把 STL 放在一個namespace 中，就定義 __STL_USE_NAMESPACES.
//  (15) 如果本程式庫由 SGI 編譯器來編譯，而且使用者並未選擇pthreads
//       或其他 threads，就定義 __STL_SGI_THREADS.
//  (16) 如果本程式庫由一個WIN32 編譯器編譯，並且在多緒模式下，就定義
//       __STL_WIN32THREADS
//  (17) 適當地定義與namespace 相關的macros 如 __STD, __STL_BEGIN_NAMESPACE。
//  (18) 適當地定義 exception 相關的macros 如 __STL_TRY, __STL_UNWIND。
//  (19) 根據 __STL_ASSERTIONS 是否定義，將 __stl_assert 定義為一個
//       測試動作或一個null macro。

#ifdef _PTHREADS
#   define __STL_PTHREADS
#endif

# if defined(__sgi) && !defined(__GNUC__)
// 使用 SGI STL 但卻不是使用 GNU C++
#   if !defined(_BOOL)
#     define __STL_NEED_BOOL
#   endif
#   if !defined(_TYPENAME_IS_KEYWORD)
#     define __STL_NEED_TYPENAME
#   endif
#   ifdef _PARTIAL_SPECIALIZATION_OF_CLASS_TEMPLATES
#     define __STL_CLASS_PARTIAL_SPECIALIZATION
#   endif
#   ifdef _MEMBER_TEMPLATES
#     define __STL_MEMBER_TEMPLATES
#   endif
#   if !defined(_EXPLICIT_IS_KEYWORD)
#     define __STL_NEED_EXPLICIT
#   endif
#   ifdef __EXCEPTIONS
#     define __STL_USE_EXCEPTIONS
#   endif
#   if (_COMPILER_VERSION >= 721) && defined(_NAMESPACES)
#     define __STL_USE_NAMESPACES
#   endif
#   if !defined(_NOTHREADS) && !defined(__STL_PTHREADS)
#     define __STL_SGI_THREADS
#   endif
# endif

// 侯捷註：撰寫《STL 源碼剖析》時，我的編譯器是 G++ 2.91.57
```

```
# ifdef __GNUC__
#   include <_G_config.h>
#   if __GNUC__ < 2 || (__GNUC__ == 2 && __GNUC_MINOR__ < 8)
#     define __STL_STATIC_TEMPLATE_MEMBER_BUG
#     define __STL_NEED_TYPENAME
#     define __STL_NEED_EXPLICIT
#   else // 這裡可看出 GNUC 2.8+ 的能力
#     define __STL_CLASS_PARTIAL_SPECIALIZATION
#     define __STL_FUNCTION_TMPL_PARTIAL_ORDER
#     define __STL_EXPLICIT_FUNCTION_TMPL_ARGS
#     define __STL_MEMBER_TEMPLATES
#   endif
    /* glibc pre 2.0 is very buggy. We have to disable thread for it.
       It should be upgraded to glibc 2.0 or later. */
#   if !defined(_NOTHREADS) && __GLIBC__ >= 2 && defined(_G_USING_THUNKS)
#     define __STL_PTHREADS
#   endif
#   ifdef __EXCEPTIONS
#     define __STL_USE_EXCEPTIONS
#   endif
# endif

# if defined(__SUNPRO_CC)
#   define __STL_NEED_BOOL
#   define __STL_NEED_TYPENAME
#   define __STL_NEED_EXPLICIT
#   define __STL_USE_EXCEPTIONS
# endif

# if defined(__COMO__)
#   define __STL_MEMBER_TEMPLATES
#   define __STL_CLASS_PARTIAL_SPECIALIZATION
#   define __STL_USE_EXCEPTIONS
#   define __STL_USE_NAMESPACES
# endif

// 侯捷註：VC6 的版本號碼是 1200
# if defined(_MSC_VER)
#   if _MSC_VER > 1000
#     include <yvals.h>     // 此檔在 MSDEV\VC98\INCLUDE
#   else
#     define __STL_NEED_BOOL
#   endif
#   define __STL_NO_DRAND48
#   define __STL_NEED_TYPENAME
#   if _MSC_VER < 1100
#     define __STL_NEED_EXPLICIT
#   endif
#   define __STL_NON_TYPE_TMPL_PARAM_BUG
```

```
#   define __SGI_STL_NO_ARROW_OPERATOR
#   ifdef _CPPUNWIND
#     define __STL_USE_EXCEPTIONS
#   endif
#   ifdef _MT
#     define __STL_WIN32THREADS
#   endif
# endif

// 侯捷註：Inprise Borland C++builder 也定義有此常數。
// C++Builder 的表現豈有如下所示這般差勁？
# if defined(__BORLANDC__)
#   define __STL_NO_DRAND48
#   define __STL_NEED_TYPENAME
#   define __STL_LIMITED_DEFAULT_TEMPLATES
#   define __SGI_STL_NO_ARROW_OPERATOR
#   define __STL_NON_TYPE_TMPL_PARAM_BUG
#   ifdef _CPPUNWIND
#     define __STL_USE_EXCEPTIONS
#   endif
#   ifdef __MT__
#     define __STL_WIN32THREADS
#   endif
# endif


# if defined(__STL_NEED_BOOL)
    typedef int bool;
#   define true 1
#   define false 0
# endif

# ifdef __STL_NEED_TYPENAME
#   define typename    // 侯捷：難道不該 #define typename class 嗎？
# endif

# ifdef __STL_NEED_EXPLICIT
#   define explicit
# endif

# ifdef __STL_EXPLICIT_FUNCTION_TMPL_ARGS
#   define __STL_NULL_TMPL_ARGS <>
# else
#   define __STL_NULL_TMPL_ARGS
# endif

# ifdef __STL_CLASS_PARTIAL_SPECIALIZATION
#   define __STL_TEMPLATE_NULL template<>
# else
```

```
#   define __STL_TEMPLATE_NULL
# endif

// __STL_NO_NAMESPACES is a hook so that users can disable namespaces
// without having to edit library headers.
# if defined(__STL_USE_NAMESPACES) && !defined(__STL_NO_NAMESPACES)
#   define __STD std
#   define __STL_BEGIN_NAMESPACE namespace std {
#   define __STL_END_NAMESPACE }
#   define  __STL_USE_NAMESPACE_FOR_RELOPS
#   define __STL_BEGIN_RELOPS_NAMESPACE namespace std {
#   define __STL_END_RELOPS_NAMESPACE }
#   define __STD_RELOPS std
# else
#   define __STD
#   define __STL_BEGIN_NAMESPACE
#   define __STL_END_NAMESPACE
#   undef  __STL_USE_NAMESPACE_FOR_RELOPS
#   define __STL_BEGIN_RELOPS_NAMESPACE
#   define __STL_END_RELOPS_NAMESPACE
#   define __STD_RELOPS
# endif

# ifdef __STL_USE_EXCEPTIONS
#   define __STL_TRY try
#   define __STL_CATCH_ALL catch(...)
#   define __STL_RETHROW throw
#   define __STL_NOTHROW throw()
#   define __STL_UNWIND(action) catch(...) { action; throw; }
# else
#   define __STL_TRY
#   define __STL_CATCH_ALL if (false)
#   define __STL_RETHROW
#   define __STL_NOTHROW
#   define __STL_UNWIND(action)
# endif

#ifdef __STL_ASSERTIONS
# include <stdio.h>
# define __stl_assert(expr) \
    if (!(expr)) { fprintf(stderr, "%s:%d STL assertion failure: %s\n", \
                __FILE__, __LINE__, # expr); abort(); }
    // 侯捷註：以上使用 stringizing operator #，詳見《多型與虛擬》第 3 章。
#else
# define __stl_assert(expr)
#endif

#endif /* __STL_CONFIG_H */
```

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