G++ 2.91.57，cygnus\cygwin-b20\include\g++\stl_numeric.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_NUMERIC_H
#define __SGI_STL_INTERNAL_NUMERIC_H

__STL_BEGIN_NAMESPACE

// 版本一
template <class InputIterator, class T>
T accumulate(InputIterator first, InputIterator last, T init) {
  for ( ; first != last; ++first)
    init = init + *first;   // 將每個元素值累加到初值 init 身上
  return init;
}

// 版本二
template <class InputIterator, class T, class BinaryOperation>
T accumulate(InputIterator first, InputIterator last, T init,
                BinaryOperation binary_op) {
```

```
  for ( ; first != last; ++first)
    init = binary_op(init, *first);  //  對每一個元素執行二元操作
  return init;
}

// 版本一
template <class InputIterator1, class InputIterator2, class T>
T inner_product(InputIterator1 first1, InputIterator1 last1,
                InputIterator2 first2, T init) {
  // 以第一序列之元素個數為據，將兩個序列都走一遍。
  for ( ; first1 != last1; ++first1, ++first2)
    init = init + (*first1 * *first2); // 執行兩個序列的一般內積
  return init;
}

// 版本二
template <class InputIterator1, class InputIterator2, class T,
          class BinaryOperation1, class BinaryOperation2>
T inner_product(InputIterator1 first1, InputIterator1 last1,
                InputIterator2 first2, T init, BinaryOperation1 binary_op1,
                BinaryOperation2 binary_op2) {
  // 以第一序列之元素個數為據，將兩個序列都走一遍。
  for ( ; first1 != last1; ++first1, ++first2)
    // 以外界提供的仿函式來取代第一版本中的 operator* 和 operator+。
    // op2 作用於兩元素間，op1 用於 op2 之結果與 init 之間。
    init = binary_op1(init, binary_op2(*first1, *first2));
  return init;
}

template <class InputIterator, class OutputIterator, class T>
OutputIterator __partial_sum(InputIterator first, InputIterator last,
                             OutputIterator result, T*) {
  T value = *first;
  while (++first != last) {
    value = value + *first;      // 前n個元素的總和
    *++result = value;           // 指定給目的端
  }
  return ++result;
}

// 版本一
template <class InputIterator, class OutputIterator>
OutputIterator partial_sum(InputIterator first, InputIterator last,
                           OutputIterator result) {
  if (first == last) return result;
  *result = *first;
  return __partial_sum(first, last, result, value_type(first));

  // 侯捷認為（並經實證），不需像上行那樣轉呼叫，可改用以下寫法（整個函式）：
  // if (first == last) return result;
  // *result = *first;
  // iterator_traits<InputIterator>::value_type value = *first;
  // while (++first != last) {
  //   value = value + *first;
  //   *++result = value;
  // }
  // return ++result;
  //
  // 這樣的觀念和作法，適用於本檔所有函式。
}

template <class InputIterator, class OutputIterator, class T,
        class BinaryOperation>
OutputIterator __partial_sum(InputIterator first, InputIterator last,
                            OutputIterator result, T*,
                            BinaryOperation binary_op) {
  T value = *first;
  while (++first != last) {
    value = binary_op(value, *first);     // 前n個元素的總計
    *++result = value;                    // 指定給目的端
  }
  return ++result;
}

// 版本二
template <class InputIterator, class OutputIterator, class
BinaryOperation>
OutputIterator partial_sum(InputIterator first, InputIterator last,
                          OutputIterator result, BinaryOperation binary_op) {
  if (first == last) return result;
  *result = *first;
  return __partial_sum(first, last, result, value_type(first), binary_op);

  // 侯捷認為（並經實證），不需像上行那樣轉呼叫，可改用以下寫法（整個函式）：
  // if (first == last) return result;
  // *result = *first;
  // iterator_trait<InputIterator>::value_type value = *first;
  // while (++first != last) {
  //   value = binary_op(value, *first);
  //   *++result = value;
  // }
  // return ++result;
  //
  // 這樣的觀念和作法，適用於本檔所有函式。
}

template <class InputIterator, class OutputIterator, class T>
OutputIterator __adjacent_difference(InputIterator first, InputIterator last,
                                              OutputIterator result, T*) {
  T value = *first;
  while (++first != last) {     // 走過整個範圍
    T tmp = *first;
    *++result = tmp - value;    // 將相鄰兩元素的差額（後-前），指派給目的端
    value = tmp;
  }
  return ++result;
}

// 版本一
template <class InputIterator, class OutputIterator>
OutputIterator adjacent_difference(InputIterator first, InputIterator last,
                                   OutputIterator result) {
  if (first == last) return result;
  *result = *first;    // 首先記錄第一個元素
  return __adjacent_difference(first, last, result, value_type(first));
}

template <class InputIterator, class OutputIterator, class T,
         class BinaryOperation>
OutputIterator __adjacent_difference(InputIterator first, InputIterator last,
                                     OutputIterator result, T*,
                                     BinaryOperation binary_op) {
  T value = *first;
  while (++first != last) {     // 走過整個範圍
    T tmp = *first;
    *++result = binary_op(tmp, value); // 將相鄰兩元素的運算結果，指派給目的端
    value = tmp;
  }
  return ++result;
}

// 版本二
template <class InputIterator, class OutputIterator, class
BinaryOperation>
OutputIterator adjacent_difference(InputIterator first, InputIterator last,
                                   OutputIterator result,
                                   BinaryOperation binary_op) {
  if (first == last) return result;
  *result = *first;    // 首先記錄第一個元素
  return __adjacent_difference(first, last, result, value_type(first),
                               binary_op);
}

// 版本二，冪次方。如果指定為乘法運算，則當n >= 0 時傳回 x ** n。
// 注意，"multiplication" 必須滿足結合律（associative），
//  但不需滿足交換律（commutative）。
template <class T, class Integer, class MonoidOperation>
```

```
T power(T x, Integer n, MonoidOperation op) {
  if (n == 0)
    return identity_element(op);     // 取出「證同元素」identity element.
  else {
    while ((n & 1) == 0) {
      n >>= 1;
      x = op(x, x);
    }

    T result = x;
    n >>= 1;
    while (n != 0) {
      x = op(x, x);
      if ((n & 1) != 0)
        result = op(result, x);
      n >>= 1;
    }
    return result;
  }
}

// 版本一，乘冪。
template <class T, class Integer>
inline T power(T x, Integer n) {
  return power(x, n, multiplies<T>());
}

// 侯捷：iota 是什麼的縮寫？
// 函式意義：在 [first,last) 範圍內填入value, value+1, value+2...。
template <class ForwardIterator, class T>
void iota(ForwardIterator first, ForwardIterator last, T value) {
  while (first != last) *first++ = value++;
}

__STL_END_NAMESPACE

#endif /* __SGI_STL_INTERNAL_NUMERIC_H */

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