G++ 2.91.57，cygnus\cygwin-b20\include\g++\stl_heap.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 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_HEAP_H
#define __SGI_STL_INTERNAL_HEAP_H

__STL_BEGIN_NAMESPACE

#if defined(__sgi) && !defined(__GNUC__) && (_MIPS_SIM != _MIPS_SIM_ABI32)
#pragma set woff 1209
#endif

// 以下這組 push_back()不允許指定「大小比較標準」
template <class RandomAccessIterator, class Distance, class T>
void __push_heap(RandomAccessIterator first, Distance holeIndex,
              Distance topIndex, T value) {
  Distance parent = (holeIndex - 1) / 2; // 找出父節點
  while (holeIndex > topIndex && *(first + parent) < value) {
    // 當尚未到達頂端，且父節點小於新值（於是不符合 heap 的次序特性）
    // 由於以上使用 operator<，可知 STL heap 是一種 max-heap（大者為父）。
    *(first + holeIndex) = *(first + parent); // 令洞值為父值
    holeIndex = parent; // percolate up：調整洞號，向上提昇至父節點。
    parent = (holeIndex - 1) / 2;     // 新洞的父節點
  }    // 持續至頂端，或滿足 heap 的次序特性為止。
  *(first + holeIndex) = value; // 令洞值為新值，完成安插動作。
}

template <class RandomAccessIterator, class Distance, class T>
inline void __push_heap_aux(RandomAccessIterator first,
                            RandomAccessIterator last, Distance*, T*) {
  __push_heap(first, Distance((last - first) - 1), Distance(0),
              T(*(last - 1)));
  // 以上係根據 implicit representation heap 的結構特性：新值必置於底層
  // 容器的最尾端，此即第一個洞號：(last-first)–1。
}

template <class RandomAccessIterator>
inline void push_heap(RandomAccessIterator first, RandomAccessIterator last) {
  // 注意，此函式被呼叫時，新元素應已置於底層容器的最尾端。
  __push_heap_aux(first, last, distance_type(first), value_type(first));
}

// 以下這組 push_back()允許指定「大小比較標準」
template <class RandomAccessIterator, class Distance, class T, class Compare>
void __push_heap(RandomAccessIterator first, Distance holeIndex,
                 Distance topIndex, T value, Compare comp) {
  Distance parent = (holeIndex - 1) / 2;
  while (holeIndex > topIndex && comp(*(first + parent), value)) {
    *(first + holeIndex) = *(first + parent);
    holeIndex = parent;
    parent = (holeIndex - 1) / 2;
  }
  *(first + holeIndex) = value;
}

template <class RandomAccessIterator, class Compare, class Distance, class T>
inline void __push_heap_aux(RandomAccessIterator first,
                            RandomAccessIterator last, Compare comp,
                            Distance*, T*) {
  __push_heap(first, Distance((last - first) - 1), Distance(0),
              T(*(last - 1)), comp);
}

template <class RandomAccessIterator, class Compare>
inline void push_heap(RandomAccessIterator first, RandomAccessIterator last,
                      Compare comp) {
  __push_heap_aux(first, last, comp, distance_type(first), value_type(first));
}

// 以下這個 __adjust_heap() 不允許指定「大小比較標準」
template <class RandomAccessIterator, class Distance, class T>
```

```
void __adjust_heap(RandomAccessIterator first, Distance holeIndex,
                   Distance len, T value) {
  Distance topIndex = holeIndex;
  Distance secondChild = 2 * holeIndex + 2;   // 洞節點之右子節點
  while (secondChild < len) {
    // 比較洞節點之左右兩個子值，然後以 secondChild 代表較大子節點。
    if (*(first + secondChild) < *(first + (secondChild - 1)))
      secondChild--;
    // Percolate down：令較大子值為洞值，再令洞號下移至較大子節點處。
    *(first + holeIndex) = *(first + secondChild);
    holeIndex = secondChild;
    // 找出新洞節點的右子節點
    secondChild = 2 * (secondChild + 1);
  }
  if (secondChild == len) { // 沒有右子節點，只有左子節點
    // Percolate down：令左子值為洞值，再令洞號下移至左子節點處。
    *(first + holeIndex) = *(first + (secondChild - 1));
    holeIndex = secondChild - 1;
  }
  // 將欲調整值填入目前的洞號內。注意，此時肯定滿足次序特性。
  // 依侯捷之見，下面直接改為 *(first + holeIndex) = value; 應該可以。
  __push_heap(first, holeIndex, topIndex, value);
}

// 以下這組 __pop_heap() 不允許指定「大小比較標準」
template <class RandomAccessIterator, class T, class Distance>
inline void __pop_heap(RandomAccessIterator first, RandomAccessIterator last,
                       RandomAccessIterator result, T value, Distance*) {
  *result = *first; // 設定尾值為首值，於是尾值即為欲求結果，
                    // 可由客端稍後再以底層容器之 pop_back() 取出尾值。
  __adjust_heap(first, Distance(0), Distance(last - first), value);
  // 以上欲重新調整 heap，洞號為 0（亦即樹根處），欲調整值為 value（原尾值）。
}

template <class RandomAccessIterator, class T>
inline void __pop_heap_aux(RandomAccessIterator first,
                           RandomAccessIterator last, T*) {
  __pop_heap(first, last-1, last-1, T(*(last-1)), distance_type(first));
  // 以上，根據 implicit representation heap 的次序特性，pop動作的結果
  // 應為底層容器的第一個元素。因此，首先設定欲調整值為尾值，然後將首值調至
  // 尾節點（所以以上將迭代器result設為last-1）。然後重整 [first, last-1)，
  // 使之重新成一個合格的 heap。
}

template <class RandomAccessIterator>
inline void pop_heap(RandomAccessIterator first, RandomAccessIterator last) {
  __pop_heap_aux(first, last, value_type(first));
}
```

```
// 以下這個 __adjust_heap() 允許指定「大小比較標準」
template <class RandomAccessIterator, class Distance, class T, class Compare>
void __adjust_heap(RandomAccessIterator first, Distance holeIndex,
                Distance len, T value, Compare comp) {
  Distance topIndex = holeIndex;
  Distance secondChild = 2 * holeIndex + 2;
  while (secondChild < len) {
    if (comp(*(first + secondChild), *(first + (secondChild - 1))))
      secondChild--;
    *(first + holeIndex) = *(first + secondChild);
    holeIndex = secondChild;
    secondChild = 2 * (secondChild + 1);
  }
  if (secondChild == len) {
    *(first + holeIndex) = *(first + (secondChild - 1));
    holeIndex = secondChild - 1;
  }
  __push_heap(first, holeIndex, topIndex, value, comp);
}

// 以下這組 __pop_heap() 允許指定「大小比較標準」
template <class RandomAccessIterator, class T, class Compare, class Distance>
inline void __pop_heap(RandomAccessIterator first, RandomAccessIterator last,
                   RandomAccessIterator result, T value, Compare comp,
                   Distance*) {
  *result = *first;
  __adjust_heap(first, Distance(0), Distance(last - first), value, comp);
}

template <class RandomAccessIterator, class T, class Compare>
inline void __pop_heap_aux(RandomAccessIterator first,
                       RandomAccessIterator last, T*, Compare comp) {
  __pop_heap(first, last - 1, last - 1, T(*(last - 1)), comp,
           distance_type(first));
}

template <class RandomAccessIterator, class Compare>
inline void pop_heap(RandomAccessIterator first, RandomAccessIterator last,
                 Compare comp) {
    __pop_heap_aux(first, last, value_type(first), comp);
}

// 以下這組 make_heap() 不允許指定「大小比較標準」。
template <class RandomAccessIterator, class T, class Distance>
void __make_heap(RandomAccessIterator first, RandomAccessIterator last, T*,
               Distance*) {
  if (last - first < 2) return; // 如果長度為 0 或 1，不必重新排列。
  Distance len = last - first;
  // 找出第一個需要重排的子樹頭部，以 parent 標示出。由於任何葉節點都不需執行
```

```
    // perlocate down，所以有以下計算。parent 命名不佳，名為 holeIndex 更好。
    Distance parent = (len - 2)/2;

    while (true) {
      // 重排以 parent 為首的子樹。len 是為了讓 __adjust_heap() 判斷操作範圍
      __adjust_heap(first, parent, len, T(*(first + parent)));
      if (parent == 0) return;     // 走完根節點，就結束。
      parent--;                    // （即將重排之子樹的）頭部向前一個節點
    }
  }

  // 將 [first,last) 排列為一個 heap。
  template <class RandomAccessIterator>
  inline void make_heap(RandomAccessIterator first, RandomAccessIterator last) {
    __make_heap(first, last, value_type(first), distance_type(first));
  }

  // 以下這組 make_heap() 允許指定「大小比較標準」。
  template <class RandomAccessIterator, class Compare, class T, class Distance>
  void __make_heap(RandomAccessIterator first, RandomAccessIterator last,
                   Compare comp, T*, Distance*) {
    if (last - first < 2) return;
    Distance len = last - first;
    Distance parent = (len - 2)/2;

    while (true) {
      __adjust_heap(first, parent, len, T(*(first + parent)), comp);
      if (parent == 0) return;
      parent--;
    }
  }

  template <class RandomAccessIterator, class Compare>
  inline void make_heap(RandomAccessIterator first, RandomAccessIterator last,
                        Compare comp) {
    __make_heap(first, last, comp, value_type(first), distance_type(first));
  }

  // 以下這個 sort_heap() 不允許指定「大小比較標準」
  template <class RandomAccessIterator>
  void sort_heap(RandomAccessIterator first, RandomAccessIterator last) {
    // 以下，每執行一次 pop_heap()，極值（在STL heap中為極大值）即被放在尾端。
    // 扣除尾端再執行一次 pop_heap()，次極值又被放在新尾端。一直下去，最後即得
    // 排序結果。
    while (last - first > 1)
       pop_heap(first, last--); // 每執行 pop_heap() 一次，操作範圍即退縮一格。
  }

  // 以下這個 sort_heap() 允許指定「大小比較標準」
```

```
template <class RandomAccessIterator, class Compare>
void sort_heap(RandomAccessIterator first, RandomAccessIterator last,
               Compare comp) {
  while (last - first > 1)
    pop_heap(first, last--, comp);
}

#if defined(__sgi) && !defined(__GNUC__) && (_MIPS_SIM != _MIPS_SIM_ABI32)
#pragma reset woff 1209
#endif

__STL_END_NAMESPACE

#endif /* __SGI_STL_INTERNAL_HEAP_H */

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