

```
1 //HeapSort.h
2 //Template functions to perform a heap sort in ascending order
3 //Programmer: Randy Miller
4 //Date last modified: April 8, 2015
5
6 /*For an exposition of this code and the data structure that it implements, cf.
7 http://www.personal.psu.edu/jum385/pdfs/HeapSort.pdf */
8
9
10 #ifndef HEAPSORT_H
11 #define HEAPSORT_H
12
13 #include <algorithm>
14
15 /*In the following code the typename 'I' represents a random-access iterator.
16 Besides supporting an iterator of this category, the type of the underlying
17 container template must be completely ordered by a (possibly overloaded) operator
18 '<', according to which this algorithm will sort in an ascending order.*/
19
20 template<typename I>
21 void heapSort(I lower, I upper);
22
23 template<typename I>
24 void makeHeap(I lower, I upper);
25
26 template<typename I>
27 void undoHeap(I lower, I upper);
28
29 template<typename I>
30 void insertIntoHeap(I lower, I upper);
31
32 template<typename I>
33 void removeFromHeap(I lower, I upper);
34
35 template<typename I>
```

```
36 void descendThruHeap(I lower, I upper, I mover);
37
38
39 //sort the sequence
40 template<typename I>
41 void heapSort(I lower, I upper)
42 {
43     makeHeap(lower, upper);
44     undoHeap(lower, upper);
45 }
46
47 //create a heap in the range >= lower and < upper
48 template<typename I>
49 void makeHeap(I lower, I upper)
50 {
51     if (--upper == lower)
52         return;
53     else
54     {
55         makeHeap(lower, upper);
56         insertIntoHeap(lower, upper);
57     }
58 }
59
60 //replace a heap in the range >= lower and < upper with a sorted sequence
61 template<typename I>
62 void undoHeap(I lower, I upper)
63 {
64     if (--upper == lower)
65         return;
66     else
67     {
68         removeFromHeap(lower, upper);
69         undoHeap(lower, upper);
70     }
```

```
71 }
72
73 //insert the element at upper into a heap in the range >= lower and < upper
74 template<typename I>
75 void insertIntoHeap(I lower, I upper)
76 {
77     I child = upper,
78     parent = lower + ((child - lower) - 1) / 2;
79     if (parent < lower || child == parent || *child <= *parent)
80         return;
81     else
82     {
83         std::iter_swap(child, parent);
84         insertIntoHeap(lower, parent);
85     }
86 }
87
88 /*remove the element now at lower to the front of the sorted sequence at upper,
89 and restore the properties of a heap in the range >= lower and < upper*/
90 template<typename I>
91 void removeFromHeap(I lower, I upper)
92 {
93     std::iter_swap(lower, upper);
94     descendThruHeap(lower, upper, lower);
95 }
96
97 /*reposition the element now at mover to its proper place in a heap
98 in the range >= lower and < upper*/
99 template<typename I>
100 void descendThruHeap(I lower, I upper, I mover)
101 {
102     I parent = mover,
103     child1 = (((2 * (parent - lower) + 1) <= (upper - lower)) ?
104             (lower + 2 * (parent - lower) + 1) : upper);
105     if (child1 == upper)
```

```
106     return;
107     I   child2 = child1 + 1,
108         childx = ((child2 == upper || *child1 >= *child2) ? child1 : child2);
109     if (*parent >= *childx)
110         return;
111     else
112     {
113         std::iter_swap(parent, childx);
114         descendThruHeap(lower, upper, childx);
115     }
116 }
117
118 #endif
```