

Program 4

Due: April 3, 2009 (by midnight).

For this programming assignment you will modify the **BinarySearchTree** template class from Program 2 to create a mergeable **BinaryHeap** template class that does not use an array to store elements, but uses the pointer-based binary tree implementation used in the **BinarySearchTree** class. The details follow.

1. You can start with your solution to Program 2 or mine. The main change for the **BinaryHeap** class is to maintain the heap property (parent \leq children), rather than the binary search tree property. Your **BinaryHeap** class will provide the following methods (all other methods should be removed, unless you want to change them to work correctly with the heap).
 - a. Constructor, destructor, **makeEmpty**, **isEmpty** – Should be the same as before.
 - b. **insert** – Inserts a new element into the heap while maintaining the heap property. Try to keep lower elements closer to the root.
 - c. **deleteMin** – Removes the minimum element from the heap and reorganizes the heap to maintain the heap property. This method should take a reference argument that is set to the minimum element.
 - d. **merge** – Merges the given binary heap with the one passed in as an argument.
 - e. **inOrder**, **preOrder**, **postOrder** and **printTraversals** – Should be the same as before from Program 2. Note that the in-order traversal will not necessarily print the elements in order.
2. You will test your implementation by building three different binary heaps containing integers. The first heap H1 will insert the ten integers 1-10 in order (1, 2, 3, ..., 10) into an empty binary heap. The second heap H2 will insert the ten integers 11-20 in reverse order (20, 19, 18, ..., 11) into an empty binary heap. The third heap H3 should be initially empty, then heap H1 should be merged into H3, and then heap H2 should be merged into H3. For each of these three heaps, your main program should build the heap as specified, and then call **PrintTraversals()**. Next, your main program should call **deleteMin** twenty times on H3, calling **PrintTraversals()** after each call to **deleteMin**. So the output will be similar that shown on the next page.
3. Create a **readme.txt** file that describes exactly how to compile and execute your program and on what platform.
4. Collect your source code, readme file and any other files needed to compile and execute your program into one ZIP file called **<your_last_name>-pgm4.zip** and include it as an attachment to an email to me (holder@wsu.edu) by the deadline. Grading will be based not only on correctness, but also on programming style and documentation.

Output format:

Heap 1:

Pre-Order: ...
Post-Order: ...
In-Order: ...

Heap 2:

Pre-Order: ...
Post-Order: ...
In-Order: ...

Heap 3:

Pre-Order: ...
Post-Order: ...
In-Order: ...

Performed deleteMin on Heap 3:

Pre-Order: ...
Post-Order: ...
In-Order: ...

Performed deleteMin on Heap 3:

Pre-Order: ...
Post-Order: ...
In-Order: ...

.
. .
. .