CS 474: Object Oriented Programming Languages and Environments
Fall 2012

First Smallalk project

Due time: 9:00 pm on Wednesday 10/31/2012

You are required to create a simple *Heap calculator* program in Cincom Smalltalk while taking advantage of Smalltalk's automatic GUI builder. The calculator allows an interactive user to enter and edit information about two sets containing integer numbers and implemented as binary heaps. The calculator supports basic set operations such as union, intersection and difference between the two heaps. *For efficiency reasons, you are required to implement each heap as a tree, not as an array.*

In short, a binary heap is a binary tree subject to two properties: (1) Each level in the tree (except possibly for the last) is fully filled, meaning that each node has two children; and (2) each node is greater than its two children. Your binary heaps must maintain the heap property at all times. Also, you must implement heaps using a *Node* structure with two children, left child and a right child.

At any point in time, the user will edit and modify one the two binary heaps, that is, H_1. When the editing is complete, the user will issue a command that will "heapify" the set, that is, create a binary heap out of the numbers entered by a user. An additional command, allows an interactive user to save H_1 as the second heap H_2. This operation will copy the first heap (H_1) into the second heap; however, H_1 is not modified. The two heap structures share no data structures. Your binary heap will support also heap operations such as union, intersection and difference. Set operations take place between H_1 and H_2; operation results are always stored in H_1. The previous contents of H_1 are lost.

The GUI of your program should support the following functionality; you should choose an appropriate Smalltalk widget to implement each piece of functionality.

1. *Clear heap.* This function allows interactive users to delete the current H_1 heap. The previous value stored in H_1 is lost.

2. *Switch heaps.* The heaps associated with H_1 and H_2 are swapped, meaning that H_1 will receive the previous H_2 heap and vice versa.

3. *Save heap.* The H_1 is copied into H_2. The previous content of H_2 is lost. The content of H_1 is not affected. The two heaps must not share any data structures, that is, they can be modified independently of each other.

4. *Display heap contents.* The numeric values stored in the two heaps are displayed in an appropriate widget or widgets. For each heap, display in breadth-first order the value stored in each node, and the values of the two children of the node. The two heaps are not modified.

5. *Add element.* This function allows a user to add a new integer to H_1. The value is read from an appropriate *line input* widget. No action is taken if the number in question is already in the heap. The insertion should preserve the binary heap properties of H_1.

6. *Remove element.* This function allows a user to remove an element from H_1. The value is entered from an appropriate *line input* widget. No action is taken if the number in question is not in the heap. Otherwise, the nodes should be rearranged in such a way as to preserve the properties of a heap in the resulting tree.

7. *Union.* This element takes the set union of H_1 and H_2; it stores the resulting value in H_1. The previous content of H_1 is lost. H_2 is not modified by this operation.

8. *Intersection.* This element takes the set intersection of H_1 and H_2; it stores the resulting value in H_1. The previous content of H_1 is lost. H_2 is not modified by this operation.

9. *Difference.* This element takes the set difference $H_2 - H_1$ and stores the result in heap H_1. The previous content of H_1 is lost. H_2 is not modified by this operation.

Now you must implement a *Tertiary Heap* data structure, as a subclass of your *Binary Heap* class. Tertiary heaps are distinguished from binary heaps in that each node has exactly three children. This is the only difference with respect to a binary heap. In particular, the two heap properties above still hold in the case of a tertiary heap. Your interface should allow users to switch between the binary heap and tertiary heap implementation for H_1 and H_2 using an appropriate widget. When switching from one to the other implementation, H_1 and H_2 are cleared. An interactive user will then start anew with the chosen implementation. Make sure that your inheritance scheme is well formed; do not duplicate in the subclass functionality or implementation inherited from the superclass.