Worksheet 35: Skew Heaps

In Preparation: Read Chapter 11 to learn more about the Priority queue data type. If you have not done so already, complete worksheet 33 on the Heap data type.

For any node in a heap the relative order of the left and right children is unimportant. The skew heap builds on this property, and results in a very different organization from the traditional heap. The skew heap makes two observations. First, left and right children can always be exchanged with each other, since their order is unimportant. Second, both insertions and removals from a heap can be implemented as special cases of a more general task to merge two heaps into one.

It is easy to see how the remove is similar to a merge. When the smallest (that is, root) element is removed, you are left with two trees, namely the left and right child trees. To build a new heap you can simply merge the two. To view addition as a merge, consider the existing heap as one argument, and a tree with only the single new node as the second. Merge the two to produce the new heap.

A skew heap makes no attempt to guarantee the balance of its internal tree. Potentially, this means that a tree could become thin and unbalanced. But this is where the first observation is used. During the merge process the left and right children are systematically swapped. The result is that a thin and unbalanced tree cannot remain so. It can be shown (although the details are complex and not presented here) that amortized over time, each operation in a skew heap is no worst than $O(\log n)$.

The following illustrates the addition of the value 10 to an existing tree. Notice how a tree with a long right path becomes a tree with a long left path.
The merge algorithm for a skew heap can be described as follows:

Node merge (Node left, Node right)
   if (left is null) return right
   if (right is null) return left
   if (left child value < right child value) {
      Node temp = left.left;
      left.left = merge(left.right, right)
      left.right = temp
      return left;
   } else {
      Node temp = right.right
      right.right = merge(right.left, left)
      right.left = temp
      return right
   }

Complete the implementation of the SkewHeap based on these ideas. The only function you need to implement is merge.

Struct skewHeap {
   struct node * root;
};

void skewHeapInit (struct skewHeap * sk) { sk->root = 0; }

void skewHeapAdd (struct skewHeap *sk) {
   struct node *n = (struct node *) malloc(sizeof(struct node));
   assert(n != 0);  n->value = 0; n->leftchild = 0; n->rightchild = 0;
   s->root = skewHeapMerge(s->root, n);
}

EleType skewHeapGetFirst (struct skewHeap *sk) {
   assert (sk->root != 0); return sk->root->value; }

void skewHeapRemoveFirst (struct skewHeap *sk) {
   struct node * n = sk->root; free(n);
   sk->root = skewHeapMerge(n->leftchild, n->rightchild);
}

stuct node * skewHeapMerge (struct node * left, struct node *right) {
}