Depending on the kind of node to be deleted a node can be removed in one of three ways. Remember, we wish to preserve the order property enforced on binary trees. The order property is the left child is less than the parent and the right child is greater than or equal to the parent.

1. The node is a *leaf.* Delete the node immediately.

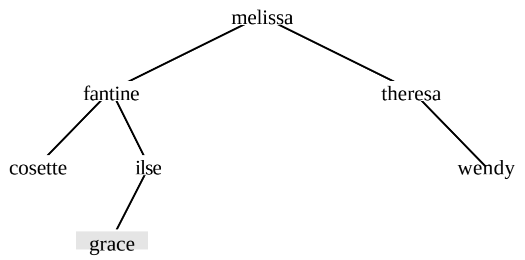

Figure 1. Before deleting a leaf (grace)

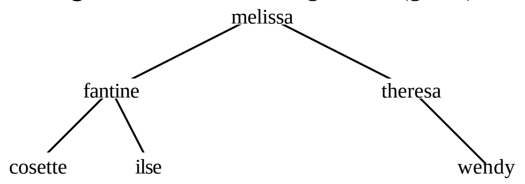

Figure 2. After deleting a leaf (grace)

2. The node has *one child.* Replace the node with its child.

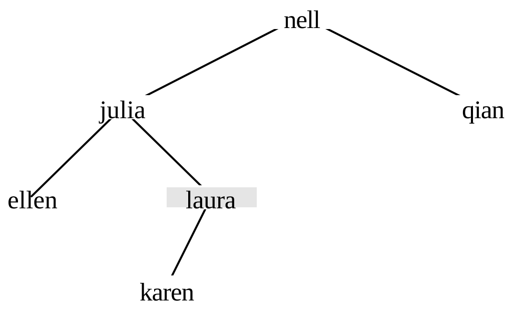

Figure 3. Before deleting a node with one child (laura)

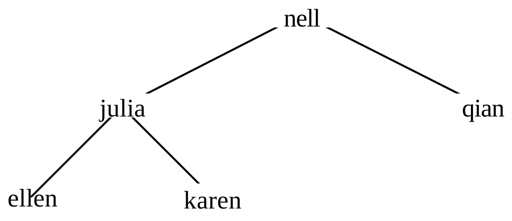

Figure 3. After deleting a node with one child (laura)

3. The node has *two children.* Recursively replace the node with the node having the smallest key in the *right* sub tree.

   Since every key in the right sub tree is greater than or (possibly) equal to the key in the parent node, the smallest key in the right sub tree will be less than or (possibly) equal to any key in the new right sub tree. Since any key in the right sub tree is greater than any key in the left sub tree, the smallest key in the right sub tree is the obvious candidate to replace the parent.

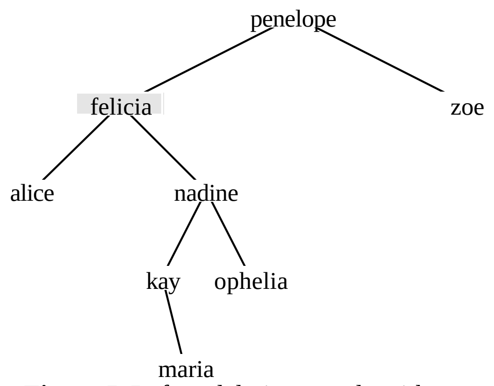

Figure 5. Before deleting a node with two children (felicia)

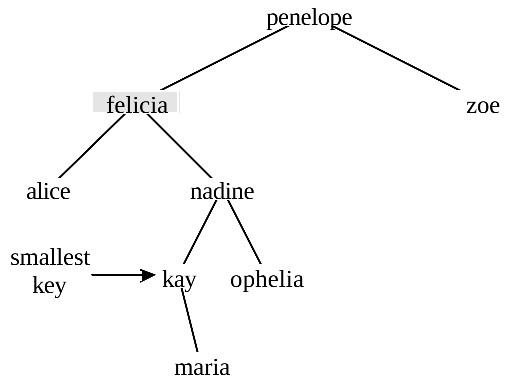

Figure 6. Identification of the smallest key in the right sub tree (kay)

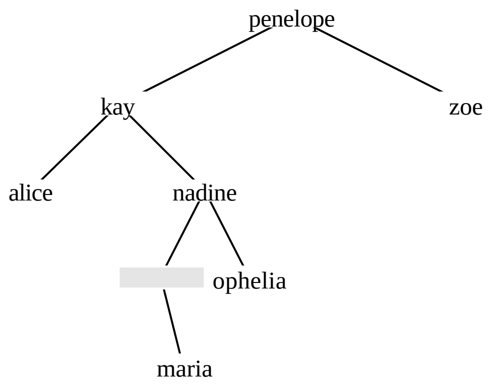

Figure 7. Replace key felicia with a key kay

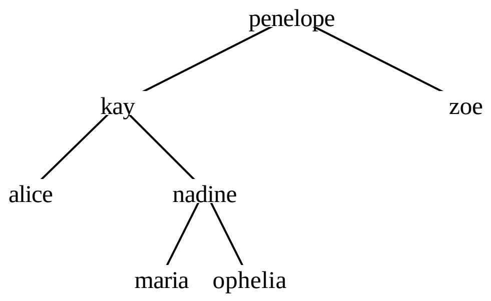

Figure 8. Delete the node whose former key was kay

```
class Tree {
      struct Node {
            Node* LNode;
            int key
            Node* RNode;
            . . .
      };
      . . .
public:
      . . .
      Node* Remove(Node* n, int key);
      . . .
};


Tree::Node* Tree::Remove(Node* n, int key)
{     if (!n) return n;
      if (key==n->key) {
      if (n->LNode && n->RNode) {                    //two children
            n->key=FindMin(n->RNode)->key;
            n->RNode=Remove(n->RNode,n->key);
            return n;
      }
      if (n->LNode) {                                //one child on the left
            Node* d=n;
            n=n->LNode;
            delete d;
            return n;
     }
     if (n->RNode) {                                 //one child on the right
            Node* d=n;
            n=n->RNode;
            delete d;
            return n;
     }
     if (n->LNode==0 && n->RNode==0) {               //no children, a leaf
            delete n;
            return 0;
     }
}
```