Binary Search Tree - Delete

```
typedef struct {
   int data;
   node *left;
   node *right;
} node;

// Deletes value from a BST rooted at root. value must be in the tree in
// to work.  Returns a pointer to the root of the resulting tree.
node* delete(node* root, int value) {
  node *delnode, *new_del_node, *save_node, *par;
  int save_val;

  delnode = _____________________; // Get a pointer to the node to delete.

  par = _________________________; // Get the parent of this node.

  // Case 1: the node to delete is a leaf node.
  if (___________________________) {

    // Deleting the only node in the tree.
    if (par == NULL) {
      _____________________; // free the memory for the node.
      return ________________;
    }

    // Deletes the node if it's a left child.
    if (_____________________) {
      free(_____________________); // Free the memory for the node.
      _____________________ = NULL;
    }

    // Deletes the node if it's a right child.
    else {
      free(_____________________); // Free the memory for the node.
      _____________________ = NULL;
    }

    return root; // Return the root of the new tree.
  }

  // Case 2: the node to be deleted only has a left child.
  if (_____________________________________) {

    // Deleting the root node of the tree.
    if (_____________________) {
      save_node = _____________________;
      free(_____________________); // Free the node to delete.
      return _________; // Return the new root node of the resulting tree.
    }

    // Deletes the node if it's a left child.
```

```
    if (_____________________________) {
      save_node = _____________________; // Save the node to delete.
      par->left = _____________________; // Readjust the parent pointer.
      free(___________________); // Free the memory for the deleted node.
    }
    // Deletes the node if it's a right child.
    else {
      save_node = _____________________; // Save the node to delete.
      par->right = ____________________; // Readjust the parent pointer.
      free(___________________); // Free the memory for the deleted node.
    }

    return root; // Return the root of the tree after the deletion.
  }

  // Case 3: the node to be deleted only has a right child.
  if (_____________________________________) {

    // Node to delete is the root node.
    if (par == NULL) {
      save_node = _____________________;
      free(_____________________);
      return _____________________;
    }

    // Delete's the node if it is a left child.
    if (_____________________) {
      save_node = _____________________;
      par->left = _____________________;
      free(_____________________);
    }

    // Delete's the node if it is a right child.
    else {
      save_node = _____________________;
      par->right = _____________________;
      free(_____________________);
    }
    return root;
  }

  // Case 4:  The deleted node has 2 children, find the replacement node
  new_del_node = _____________________________________;
  save_val = _____________________________;

  delete(root, _____________________);  // Now, delete the proper value.

  // Restore the data to the original node to be deleted.
  delnode->data = _____________________;

  return root;
}
```