```c
/**
 * Lab Sheet -5, The implementation of binary search tree
 * and perform insertion/deletion operations.
 *
 * By: Santa Basnet
 * Everest Engineering College.
 * Date: 2022-02-07
 */

#include <malloc.h>

/**
 * Binary Search Tree Representation.
 */
struct BinaryTree {
    int key, record;
    struct BinaryTree *leftChild, *rightChild;
};

/**
 * Create a BST Node.
 */
struct BinaryTree *createBSTNode(int key, int record) {
    struct BinaryTree *treePointer = (struct BinaryTree *)
malloc(sizeof(BinaryTree));
    treePointer->key = key;
    treePointer->record = record;
    treePointer->leftChild = treePointer->rightChild = nullptr;
    return treePointer;
}

/**
 * Inorder Traversal.
 * 1. Visit Left Child.
 * 2. Display Data.
 * 3. visit Right Child.
 * @param rootPointer
 */
static struct BinaryTree *inOrderTraversal(struct BinaryTree *root) {
    struct BinaryTree *traverse = root;
    if (traverse != nullptr) {
        inOrderTraversal(traverse->leftChild);
        printf("\t%4d -> %d\n", traverse->key, traverse->record);
        inOrderTraversal(traverse->rightChild);
    }
    return root;
}

/**
```

```
 * Perform Insert Operation with given key and record.
 * tree, a root pointer.
 */
struct BinaryTree *SInsert(struct BinaryTree *tree, int key, int record)
{
    if (tree == nullptr) return createBSTNode(key, record);
    else if (key == tree->key) return tree;
    else if (key < tree->key) tree->leftChild = SInsert(tree->leftChild,
key, record);
    else tree->rightChild = SInsert(tree->rightChild, key, record);
    return tree;
}

/**
 * Find minimum of two nodes.
 */
static struct BinaryTree *minimumOf(struct BinaryTree *givenNode) {
    struct BinaryTree *currentNode = givenNode;
    while (currentNode && currentNode->leftChild != nullptr) currentNode
= currentNode->leftChild;
    return currentNode;
}

void printDeleted(struct BinaryTree *node) {
    if (node == nullptr) printf("\nDeleted: %d.\n", 0);
    else printf("\nDeleted: %d(%d).\n", node->key, node->record);
}

struct BinaryTree *deleteNode(struct BinaryTree *tree, int key) {
    if (tree == nullptr) return tree;
    else if (tree->key < key) tree->rightChild = deleteNode(tree-
>rightChild, key);
    else if (tree->key > key) tree->leftChild = deleteNode(tree-
>leftChild, key);
    else {
        if (tree->leftChild == nullptr && tree->rightChild == nullptr) {
            printDeleted(tree);
            free(tree);
            return nullptr;
        } else if (tree->leftChild == nullptr) {
            struct BinaryTree *tempNode = tree->rightChild;
            printDeleted(tree);
            free(tree);
            return tempNode;
        } else if (tree->rightChild == nullptr) {
            struct BinaryTree *tempNode = tree->leftChild;
            printDeleted(tree);
            free(tree);
            return tempNode;
```

```
        } else {
            struct BinaryTree *tTree = minimumOf(tree->rightChild);
            tree->key = tTree->key;
            tree->record = tTree->record;
            tree->rightChild = deleteNode(tree->rightChild, tTree->key);
        }
    }
    return tree;
}

/**
 * Perform delete operation with given key.
 */
void SDelete(struct BinaryTree *tree, int key) {
    if (tree == nullptr) printDeleted(nullptr);
    else deleteNode(tree, key);
}

/**
 * Perform delete operation of range key1 to key2.
 */
void Delete(struct BinaryTree *tree, int key1, int key2) {
    for (int i = key1; i <= key2; i++) SDelete(tree, i);
}

/**
 * Initialization of binary search tree.
 * @return aRootNode
 */
struct BinaryTree *initialize() {
    const int SIZE = 10;
    int keys[SIZE] = {
            78, 23, 79, 100, 1,
            87, 66, 33, 699, 7
    };
    int records[SIZE] = {
            7800, 2300, 7900, 10000, 100,
            8700, 6600, 3300, 69900, 700
    };

    /**
     * Declare Binary Tree Variable.
     */
    struct BinaryTree *tree = nullptr;

    /**
     * Perform insert operation.
     */
    for (int index = 0; index < SIZE; index++)
```

```
        tree = SInsert(tree, keys[index], records[index]);

    inOrderTraversal(tree);
    SDelete(tree, 79);

    inOrderTraversal(tree);

    return tree;
}

/**
 * Main program starts here.
 * @return 0 as a successful completion.
 */
int main() {
    initialize();
    return 0;
}


Output:
--------------------------------------------------------------------------
         1 -> 100
         7 -> 700
        23 -> 2300
        33 -> 3300
        66 -> 6600
        78 -> 7800
        79 -> 7900
        87 -> 8700
       100 -> 10000
       699 -> 69900

Deleted: 79(7900).
         1 -> 100
         7 -> 700
        23 -> 2300
        33 -> 3300
        66 -> 6600
        78 -> 7800
        87 -> 8700
       100 -> 10000
       699 -> 69900
```