```c
/*
* Program : Binary Search Tree Deletion
* Language : C
*/

#include <stdio.h>
#include <stdlib.h>

struct node
{
int key;
struct node *left;
struct node *right;
};

struct node *getNewNode(int val)
{
struct node *newNode = malloc(sizeof(struct node));
newNode->key = val;
newNode->left = NULL;
newNode->right = NULL;

return newNode;
}

struct node *insert(struct node *root, int val)
{
if (root == NULL)
return getNewNode(val);
if (root->key < val)
root->right = insert(root->right, val);
else if (root->key > val)
root->left = insert(root->left, val);
```

```
return root;
}

int getRightMin(struct node *root)
{
struct node *temp = root;

// min value should be present in the left most node.
while (temp->left != NULL)
{
temp = temp->left;
}

return temp->key;
}

struct node *removeNode(struct node *root, int val)
{
/*
* If the node becomes NULL, it will return NULL
* Two possible ways which can trigger this case
* 1. If we send the empty tree. i.e root == NULL
* 2. If the given node is not present in the tree.
*/
if (root == NULL)
return NULL;
/*
* If root->key < val. val must be present in the right subtree
* So, call the above remove function with root->right
*/
if (root->key < val)
root->right = removeNode(root->right, val);
```

```
/*
* if root->key > val. val must be present in the left subtree
* So, call the above function with root->left
*/
else if (root->key > val)
root->left = removeNode(root->left, val);
/*
* This part will be executed only if the root->key == val
* The actual removal starts from here
*/
else
{
/*
* Case 1: Leaf node. Both left and right reference is NULL
* replace the node with NULL by returning NULL to the calling
pointer.
* free the node
*/
if (root->left == NULL && root->right == NULL)
{
free(root);
return NULL;
}
/*
* Case 2: Node has right child.
* replace the root node with root->right and free the right node
*/
else if (root->left == NULL)
{
struct node *temp = root->right;
free(root);
return temp;
}
```

```
/*
* Case 3: Node has left child.
* replace the node with root->left and free the left node
*/
else if (root->right == NULL)
{
struct node *temp = root->left;
free(root);
return temp;
}
/*
* Case 4: Node has both left and right children.
* Find the min value in the right subtree
* replace node value with min.
* And again call the remove function to delete the node which
has the min value.
* Since we find the min value from the right subtree call the
remove function with root->right.
*/
else
{
int rightMin = getRightMin(root->right);
root->key = rightMin;
root->right = removeNode(root->right, rightMin);
}
}

// return the actual root's address
return root;
}

/*
* it will print the tree in ascending order
```

```
*/
void inorder(struct node *root)
{
if (root == NULL)
return;
inorder(root->left);
printf("%d ", root->key);
inorder(root->right);
}

int main()
{
/*
100
/ \
50 200
/ \
150 300
*/
struct node *root = NULL;
root = insert(root, 100);
root = insert(root, 50);
root = insert(root, 200);
root = insert(root, 150);
root = insert(root, 300);

printf("Initial tree :\t");
inorder(root);
printf("\n");

/* remove leaf node 300
100
/ \
```

```
50 200
/
150
*/
root = removeNode(root, 300);
printf("After deletion of 300, the new tree :\t");
inorder(root);
printf("\n");

/* remove root node 100
150
/ \
50 200
*/
root = removeNode(root, 100);
printf("After deletion of 100, the new tree :\t");
inorder(root);
printf("\n");

return 0;
}
```