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

struct Node
{
    int data;
    struct Node *left;
    struct Node *right;
    int height;
};

typedef struct Node Node;

int max(int a, int b)
{
    return (a > b) ? a : b;
}

int height(Node *N)
{
    if (N == NULL)
        return 0;
    return N -> height;
}

Node *newNode(int ele)
{
    Node *node = (Node *)malloc(sizeof( Node ));
    node -> data = ele;
    node -> left = NULL;
    node -> right = NULL;
    node -> height = 1;
    return (node);
}

Node *rightRotate(Node *y)
{
    Node *x = y -> left;
    Node *T2 = x -> right;

    x -> right = y;
    y -> left = T2;

    y -> height = max(height(y -> left), height(y -> right)) + 1;
    x -> height = max(height(x -> left), height(x -> right)) + 1;

    return x;
}

Node *leftRotate(Node *x)
{
    Node *y = x -> right;
    Node *T2 = y -> left;
```

```
    y -> left = x;
    x -> right = T2;

    x -> height = max(height(x -> left), height(x -> right)) + 1;
    y -> height = max(height(y -> left), height(y -> right)) + 1;

    return y;
}

int getBalance(Node *N)
{
    if(N == NULL)
        return 0;
    return height(N -> left) - height(N -> right);
}

Node *Insert(Node *node, int ele)
{
    int balance;

    if(node == NULL)
        return (newNode(ele));
    if(ele < node -> data)
        node -> left = Insert(node -> left, ele);
    else if (ele > node -> data)
        node -> right = Insert(node -> right, ele);
    else
        return node;

    node -> height = 1 + max(height(node -> left), height(node -> right));

    balance = getBalance(node);

    if(balance > 1 && ele < node -> left -> data)
        return rightRotate(node);

    if(balance < -1 && ele > node -> right -> data)
        return leftRotate(node);

    if(balance > 1 && ele > node -> left -> data)
    {
        node -> left = leftRotate(node -> left);
        return rightRotate(node);
    }

    if(balance < -1 && ele < node -> right -> data)
    {
        node -> right = rightRotate(node -> right);
        return leftRotate(node);
    }

    return node;
}
```

```
Node *Create(Node *root )
{
    int num, i, ele;
    printf("\n Enter number of nodes:");
    scanf("%d", &num );
    printf("\n Enter elements:");
    for( i = 0; i < num; i++)
    {
        scanf("%d", &ele);
        root = Insert( root, ele );
    }

    return root;
}

Node *minValueNode(Node *node)
{
    Node *current = node;

    while(current -> left != NULL)
        current = current -> left;

    return current;
}

Node *Delete(Node *root, int ele)
{
    int balance;

    if(root == NULL)
    {
        printf("\nTree is empty. Please create tree\n");
        return root;
    }

    if(ele < root -> data)
        root -> left = Delete(root -> left, ele);
    else if(ele > root -> data)
        root -> right = Delete(root -> right, ele);
    else
    {
        if((root -> left == NULL) || (root -> right == NULL))
        {
            Node *temp = root -> left ? root -> left : root -> right;

            if(temp == NULL)
            {
                temp = root;
                root = NULL;
            }
            else
                *root = *temp;
            free(temp);
        }
```

```
        else
        {
            Node *temp = minValueNode(root -> right);
            root -> data = temp -> data;
            root -> right = Delete(root -> right, temp -> data);
        }
    }

    if(root == NULL)
        return root;

    root -> height = 1 + max(height(root -> left), height(root -> right));

    balance = getBalance(root);
    if(balance > 1 && getBalance(root -> left) >= 0)
        return rightRotate(root);

    if(balance > 1 && getBalance(root -> left) < 0)
    {
        root -> left = leftRotate(root -> left);
        return rightRotate(root);
    }

    if(balance < -1 && getBalance(root -> right) <= 0)
        return leftRotate(root);

    if(balance < -1 && getBalance(root -> right) > 0)
    {
        root -> right = rightRotate(root -> right);
        return leftRotate(root);
    }

    return root;
}

void Inorder(Node *root)
{
    if(root != NULL)
    {
        Inorder(root -> left);
        printf("%d ", root -> data);
        Inorder(root -> right);
    }
}

int main()
{

        Node *root = NULL;
        int ele;
        int e = 1, ch;

        while( e )
        {
```

```
                printf( "\n--------------MENU--------------\n" );
                printf( "\n\t1. Create\n\t2. Insert\n\t3. Inorder
Traversal\n\t4. Delete\n\t5. Exit\n" );
                printf( "\n--------------------------------\n" );
                printf( "\n Enter your choice:" );
                scanf( "%d", &ch );

                switch( ch )
                {
                        case 1: root = Create( root );
                             break;
                        case 2:
                                printf("\n Enter the element to insert:");
                                scanf("%d", &ele);
                                root = Insert( root, ele);
                                 break;
                        case 3: Inorder( root );
                                 break;
                        case 4:
                                printf("\n Enter the element to delete :");
                                scanf("%d", &ele);
                                root = Delete( root, ele );
                                 break;
                    case 5: e = 0;
                                 break;
                        default: printf( "\n Invalid choice \n" );
                }

        }
        return 0;
}
```