main.c

Share | Run | Output | Clear

```c
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int key, height;
    struct Node *left, *right;
} Node;
int h(Node *n) { return n ? n->height : 0; }
int max(int a, int b) { return a > b ? a : b; }
int bal(Node *n) { return n ? h(n->left) - h(n->right) : 0; }
Node* newNode(int key) {
    Node *n = malloc(sizeof(Node));
    n->key = key; n->left = n->right = NULL; n->height = 1;
    return n;
}
Node* rotR(Node *y) {
    Node *x = y->left, *T = x->right;
    x->right = y; y->left = T;
    y->height = 1 + max(h(y->left), h(y->right));
    x->height = 1 + max(h(x->left), h(x->right));
    return x;
}
Node* rotL(Node *x) {
    Node *y = x->right, *T = y->left;
    y->left = x; x->right = T;
    x->height = 1 + max(h(x->left), h(x->right));
    y->height = 1 + max(h(y->left), h(y->right));
```

```
20 30
Search 20: Found


=== Code Execution Successful ===
```

main.c

Share Run

```
    return y;
}
Node* insert(Node *n, int key) {
    if (!n) return newNode(key);
    if (key < n->key) n->left = insert(n->left, key);
    else if (key > n->key) n->right = insert(n->right, key);
    else return n;
    n->height = 1 + max(h(n->left), h(n->right));
    int b = bal(n);
    if (b > 1 && key < n->left->key) return rotR(n);
    if (b < -1 && key > n->right->key) return rotL(n);
    if (b > 1 && key > n->left->key) { n->left = rotL(n->left);
        return rotR(n); }
    if (b < -1 && key < n->right->key) { n->right = rotR(n->right);
        return rotL(n); }
    return n;
}
Node* minVal(Node *n) {
    while (n->left) n = n->left;
    return n;
}
Node* delete(Node *n, int key) {
    if (!n) return n;
    if (key < n->key) n->left = delete(n->left, key);
    else if (key > n->key) n->right = delete(n->right, key);
    else {
```

Output Clear

```
20 30
Search 20: Found


=== Code Execution Successful ===
```

main.c

Share | Run | Output | Clear

```
        if (!n->left || !n->right) {
            Node *tmp = n->left ? n->left : n->right;
            if (!tmp) tmp = n, n = NULL;
            else *n = *tmp;
            free(tmp);
        } else {
            Node *tmp = minVal(n->right);
            n->key = tmp->key;
            n->right = delete(n->right, tmp->key);
        }
    }
    if (!n) return n;
    n->height = 1 + max(h(n->left), h(n->right));
    int b = bal(n);
    if (b > 1 && bal(n->left) >= 0) return rotR(n);
    if (b > 1 && bal(n->left) < 0) { n->left = rotL(n->left); return
        rotR(n); }
    if (b < -1 && bal(n->right) <= 0) return rotL(n);
    if (b < -1 && bal(n->right) > 0) { n->right = rotR(n->right);
        return rotL(n); }
    return n;
}
Node* search(Node *n, int key) {
    if (!n || n->key == key) return n;
    return key < n->key ? search(n->left, key) : search(n->right,
        key);
```

```
20 30
Search 20: Found


=== Code Execution Successful ===
```

main.c

Share | Run | Output | Clear

```
        rotR(n); }
    if (b < -1 && bal(n->right) <= 0) return rotL(n);
    if (b < -1 && bal(n->right) > 0) { n->right = rotR(n->right);
        return rotL(n); }
    return n;
}
Node* search(Node *n, int key) {
    if (!n || n->key == key) return n;
    return key < n->key ? search(n->left, key) : search(n->right,
        key);
}
void pre(Node *n) {
    if (n) { printf("%d ", n->key); pre(n->left); pre(n->right); }
}
int main() {
    Node *root = NULL;
    root = insert(root, 30);
    root = insert(root, 10);
    root = insert(root, 20);
    root = delete(root, 10);
    pre(root);
    printf("\nSearch 20: %s\n", search(root, 20) ? "Found" : "Not
        Found");
    return 0;
}
```

```
20 30
Search 20: Found


=== Code Execution Successful ===
```