```cpp
#include<bits/stdc++.h>
using namespace std;
class Node
{
    public:
    int key;
    Node *left;
    Node *right;
    int height;
};
int max(int a, int b);
int height(Node *N)
{
    if (N == NULL)
        return 0;
    return N->height;
}
int max(int a, int b)
{
    return (a > b)? a : b;
}
Node* newNode(int key)
{
    Node* node = new Node();
    node->key = key;
    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;
```

```
61      return height(N->left) -
62              height(N->right);
63  }
64  Node* insert(Node* node, int key)
65  {
66      if (node == NULL)
67          return(newNode(key));
68      if (key < node->key)
69          node->left = insert(node->left, key);
70      else if (key > node->key)
71          node->right = insert(node->right, key);
72      else
73          return node;
74
75      node->height = 1 + max(height(node->left),
76
77                             height(node->right));
78
79      int balance = getBalance(node);
80      if (balance > 1 && key < node->left->key)
81          return rightRotate(node);
82      if (balance < -1 && key > node->right->key)
83          return leftRotate(node);
84      if (balance > 1 && key > node->left->key)
85      {
86          node->left = leftRotate(node->left);
87          return rightRotate(node);
88      }
89      if (balance < -1 && key < node->right->key)
90      {
```

```
91          node->right = rightRotate(node->right);
92          return leftRotate(node);
93      }
94      return node;
95  }
96  Node * minValueNode(Node* node)
97  {
98      Node* current = node;
99      while (current->left != NULL)
100         current = current->left;
101     return current;
102 }
103 Node* deleteNode(Node* root, int key)
104 {
105     if (root == NULL)
106         return root;
107     if ( key < root->key )
108         root->left = deleteNode(root->left, key);
109     else if( key > root->key )
110         root->right = deleteNode(root->right, key);
111     else
112     {
113         if( (root->left == NULL) ||
114             (root->right == NULL) )
115         {
116             Node *temp = root->left ?
117                          root->left :
118                          root->right;
119             if (temp == NULL)
120             {
```

```
            temp = root;
            root = NULL;
        }

        else

        *root = *temp;

        free(temp);
    }
    else
    {
        Node* temp = minValueNode(root->right);
        root->key = temp->key;
        root->right = deleteNode(root->right,
                                 temp->key);
    }
}
if (root == NULL)
return root;
root->height = 1 + max(height(root->left),
                       height(root->right));
int 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);
```

```
151         return rightRotate(root);
152     }
153     if (balance < -1 &&
154         getBalance(root->right) <= 0)
155         return leftRotate(root);
156     if (balance < -1 &&
157         getBalance(root->right) > 0)
158     {
159         root->right = rightRotate(root->right);
160         return leftRotate(root);
161     }
162     return root;
163 }
164 void preOrder(Node *root)
165 {
166     if(root != NULL)
167     {
168         cout << root->key << " ";
169         preOrder(root->left);
170         preOrder(root->right);
171     }
172 }
173 int main()
174 {
175 Node *root = NULL;
176     root = insert(root, 9);
177 
178     root = insert(root, 5);
179 
180     root = insert(root, 10);
```

```
    root = insert(root, 9);

    root = insert(root, 5);

    root = insert(root, 10);

    root = insert(root, 0);

    root = insert(root, 6);

    root = insert(root, 11);

    root = insert(root, -1);

    root = insert(root, 1);

    root = insert(root, 2);

    cout << "Preorder traversal of the "
            "constructed AVL tree is \n";
    preOrder(root);
    root = deleteNode(root, 10);

    cout << "\nPreorder traversal after"

         << " deletion of 10 \n";
    preOrder(root);
    return 0;
}

```

```
Preorder traversal of the constructed AVL tree is
9 1 0 -1 5 2 6 10 11
Preorder traversal after deletion of 10
1 0 -1 9 5 2 6 11

...Program finished with exit code 0
Press ENTER to exit console.
```