```cpp
//create and add node
#include<bits/stdc++.h>
using namespace std;

struct Node{
    int data;
    Node* left = NULL;
    Node* right = NULL;
};

Node* root = NULL;

void add(int value){
    Node* nptr = new Node;
    nptr->data = value;
    nptr->left = NULL;
    nptr->right = NULL;

    if(root==NULL)
        root=nptr;
    else{
        Node* tptr = root;
        while(true){
            if(tptr->data > value){
                //left
                if(tptr->left==NULL){
                    tptr->left = nptr;
                    break;
                }
                else tptr = tptr->left;
            }else{
                //right
                if(tptr->right==NULL){
                    tptr->right = nptr;
                    break;
                }
                else tptr = tptr->right;
            }
        }
    }
}


    void print(Node* node){
        if(node!=NULL){
            print(node->left);
            cout<<node->data<<" ";
            print(node->right);
        }
    }
int32_t main()
{
    add(20);add(4);add(12);add(30);add(24);
    print(root);
}
```

```cpp
#include<bits/stdc++.h>
using namespace std;
struct Node {
    int data;
    Node* left = NULL;
    Node* right = NULL;
};
Node* root = NULL;
void add(int value) {
    Node* nptr = new No
    nptr->data = value;
    nptr->left = NULL;
    nptr->right = NULL;
    if (root == NULL)
        root = nptr;
    else {
        Node* tptr = ro
        while (true) {
            if (tptr->d
                // left
                if (tpt
                    tpt
                    bre
                } else
                    tpt
            } else {
                // right
                if (tptr->right == NULL) {
                    tptr->right = nptr;
                    break;
                } else
                    tptr = tptr->right;
            }
        }
    }
}
Node* findMinNode(Node* root) {
    while (root->left != NULL) {
        root = root->left;
    }
    return root;
}
Node* deleteNode(Node* root, int value); // Forward declaratio
Node* deleteRoot() {
    if (root == NULL) return NULL;
    Node* newRoot = NULL;
    if (root->left == NULL) {
        newRoot = root->right;
        delete root;
    } else if (root->right == NULL) {
        newRoot = root->left;
        delete root;
    } else {
        Node* minRight = findMinNode(root->right);
        root->data = minRight->data;
        root->right = deleteNode(root->right, minRight->data);
```

```
55          root->data = minRight->data;
56          root->right = deleteNode(root->right, minRight->data);
57          newRoot = root;
58      }
59      return newRoot;
60  }
61  Node* deleteNode(Node* root, int value) {
62      if (root == NULL) return NULL;
63      if (value < root->data) {
64          root->left = deleteNode(root->left, value);
65      } else if (value > root->data) {
66          root->right = deleteNode(root->right, value);
67      } else {
68          if (root->left == NULL) {
69              Node* temp = root->right;
70              delete root;
71              return temp;
72          } else if (root->right == NULL) {
73              Node* temp = root->left;
74              delete root;
75              return temp;
76          }
77          Node* minRight = findMinNode(root->right);
78          root->data = minRight->data;
79          root->right = deleteNode(root->right, minRight->data);
80      }
81      return root;
82  }
83  void inorderTraversal(Node* root) {
84      if (root != NULL) {
85          inorderTraversal(root->left);
86          cout << root->data << " ";
87          inorderTraversal(root->right);
88      }
89  }
90  int main() {
91      add(20);add(8);add(30);add(3);add(10);add(40);add(25);
92      cout << "Original BST before deleting root node:" << endl;
93      inorderTraversal(root);
94      cout << endl;
95      root = deleteRoot();
96      cout << "BST after deleting root node:" << endl;
97      inorderTraversal(root);
98      cout << endl;
99      return 0;
00  }
```

Delete_leaf

```
                } else
                    tptr = tptr->left;
            } else {
                // right
                if (tptr->right == NULL) {
                    tptr->right = nptr;
                    break;
                } else
                    tptr = tptr->right;
            }
        }
    }
}

Node* deleteLeafNode(Node* root, int value) {
    if (root == NULL) return NULL;

    if (value < root->data) {
        root->left = deleteLeafNode(root->left, value);
    } else if (value > root->data) {
        root->right = deleteLeafNode(root->right, value);
    } else {
        // Found the node to delete
        if (root->left == NULL && root->right == NULL) {
            delete root;
            return NULL; // This indicates the node has b
        }
    }
    return root;
}
void inorderTraversal(Node* root) {
    if (root != NULL) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
}
int main() {
    add(20);
    add(8);
    add(30);
    add(4);
    add(12);
    cout << "Original BST before deleting leaf node:" << endl
    inorderTraversal(root);
    cout << endl;
    int valueToDelete = 4;
    root = deleteLeafNode(root, valueToDelete);
    cout << "BST after deleting leaf node with value ";
    cout << valueToDelete << ":" << endl;
    inorderTraversal(root);
    cout << endl;
    return 0;
```

```
38  Node* deleteNode(Node* root, int value) {
39      if (root == NULL) return NULL;
40      if (value < root->data) {
41          root->left = deleteNode(root->left, value);
42      } else if (value > root->data) {
43          root->right = deleteNode(root->right, value);
44      } else {
45          // Node to be deleted is found
46          if (root->left == NULL) {
47              Node* temp = root->right;
48              delete root;
49              return temp;
50          } else if (root->right == NULL) {
51              Node* temp = root->left;
52              delete root;
53              return temp;
54          } else {
55              Node* minRight = root->right;
56              while (minRight->left != NULL) {
57                  minRight = minRight->left;
58              }
59              root->data = minRight->data;
60              root->right = deleteNode(root->right, minRight->data);
61          }
62      }
63      return root;
64  }
65  void inorderTraversal(Node* root) {
66      if (root != NULL) {
67          inorderTraversal(root->left);
68          cout << root->data << " ";
69          inorderTraversal(root->right);
70      }
71  }
72  int main() {
73      add(20);add(8);add(30);add(4);add(12);add(25);add(32)
74      cout << "Original BST before deleting node:" << endl;
75      inorderTraversal(root);
76      cout << endl;
77      int valueToDelete = 30;
78      root = deleteNode(root, valueToDelete);
79      cout << "BST after deleting node with value ";
80      cout<< valueToDelete << ":" << endl;
81      inorderTraversal(root);
82      cout << endl;
83      return 0;
84  }
```

Delete

Parents

```
int kthSmallest(Node* root, int k) {
    stack<Node*> st;
    Node* current = root;
    int count = 0;
    while (current != NULL || !st.empty()) {
        while (current != NULL) {
            st.push(current);
            current = current->left;
        }
        current = st.top();
        st.pop();
        count++;
        if (count == k) {
            return current->data;
        }
        current = current->right;
    }
    return -1;
}
int main() {
    add(20);add(8);add(22);add(4);add(12);add(10);add(14);
    int k = 3;int kthSmallestElement = kthSmallest(root, k);
    if (kthSmallestElement != -1) {
        cout << "The " << k << "-th smallest element in the BST is: " << kthSmallestElement << endl;
    } else {
        cout << "The " << k << "-th smallest element does not exist in the BST." << endl;
    }
    return 0;
}
```

```
int findMinimumValue(Node* node) {
    while (node->left != NULL) {
        node = node->left;
    }
    return node->data;
}
int findMaximumValue(Node* node) {
    int findMaximumValue(Node* node)
        node = node->right;
    }
    return node->data;
}
void print(Node* node) {
    if (node != NULL) {
        print(node->left);
        cout << node->data << " ";
        print(node->right);
    }
}
int main() {
    add(1);add(4);add(60);add(30);add(24);
    int minimumValue = findMinimumValue(root);
    cout << "Minimum value in the tree: " << minimumValue
    int maximumValue = findMaximumValue(root);
    cout << "Maximum value in the tree: " << maximumValue
    return 0;
}
```