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

typedef struct node {
    struct node *left;
    int element;
    struct node *right;
} Node;

Node *Insert(Node *Tree, int e);
void Display(Node *Tree);
Node *Delete(Node *Tree, int e);
Node *FindMin(Node *Tree);
Node *FindMax(Node *Tree);
Node *Find(Node *Tree, int e);

Node *Insert(Node *Tree, int e) {
    if (Tree == NULL) {
        Node *NewNode = malloc(sizeof(Node));
        NewNode->element = e;
        NewNode->left = NULL;
        NewNode->right = NULL;
        Tree = NewNode;
    } else if (e < Tree->element) {
        Tree->left = Insert(Tree->left, e);
    } else if (e > Tree->element) {
        Tree->right = Insert(Tree->right, e);
    }
    return Tree;
}
```

```
void Display(Node *Tree) {
    if (Tree != NULL) {
        Display(Tree->left);
        printf("%d\t", Tree->element);
        Display(Tree->right);
    }
}

Node *Delete(Node *Tree, int e) {
    if (Tree == NULL) {
        return NULL;
    }
    if (e < Tree->element) {
        Tree->left = Delete(Tree->left, e);
    } else if (e > Tree->element) {
        Tree->right = Delete(Tree->right, e);
    } else {
        Node *TempNode;
        if (Tree->left && Tree->right) {
            TempNode = FindMin(Tree->right);
            Tree->element = TempNode->element;
            Tree->right = Delete(Tree->right, Tree->element);
        } else {
            TempNode = Tree;
            if (Tree->left == NULL) {
                Tree = Tree->right;
            } else if (Tree->right == NULL) {
                Tree = Tree->left;
            }
```

```
            free(TempNode);
        }
    }
    return Tree;
}

Node *FindMin(Node *Tree) {
    if (Tree != NULL) {
        while (Tree->left != NULL) {
            Tree = Tree->left;
        }
        return Tree;
    }
    return NULL;
}

Node *FindMax(Node *Tree) {
    if (Tree == NULL) {
        return NULL;
    } else if (Tree->right == NULL) {
        return Tree;
    } else {
        return FindMax(Tree->right);
    }
}

Node *Find(Node *Tree, int e) {
    if (Tree == NULL) {
        return NULL;
    } else if (e < Tree->element) {
```

```
        return Find(Tree->left, e);
    } else if (e > Tree->element) {
        return Find(Tree->right, e);
    } else {
        return Tree;
    }
}

int main() {
    Node *Tree = NULL;
    Node *Result = NULL;
    int n, i, e, ch;
    printf("Enter number of nodes in the tree : ");
    scanf("%d", &n);
    printf("Enter the elements :\n");
    for (i = 1; i <= n; i++) {
        scanf("%d", &e);
        Tree = Insert(Tree, e);
    }

    do {
        printf("1. Insert \n2. Find \n3. Find Min \n4. Find Max \n5. Delete \n6. Display \n7. Exit\n");
        printf("Enter your choice : ");
        scanf("%d", &ch);
        switch (ch) {
            case 1:
                printf("Enter the element :");
                scanf("%d", &e);
                Tree = Insert(Tree, e);
                break;
```

```
case 2:
    printf("Enter the element to find : ");
    scanf("%d", &e);
    Result = Find(Tree, e);
    if (Result == NULL)
        printf("Element is not found...!\n");
    else
        printf("Element is found...!\n");
    break;
case 3:
    Result = FindMin(Tree);
    if (Result == NULL)
        printf("Tree is empty...!\n");
    else
        printf("Min element is: %d\n", Result->element);
    break;
case 4:
    Result = FindMax(Tree);
    if (Result == NULL)
        printf("Tree is empty...!\n");
    else
        printf("Max element is: %d\n", Result->element);
    break;
case 5:
    printf("\nEnter the element to delete : \n");
    scanf("%d", &e);
    Tree = Delete(Tree, e);
    printf("Tree elements in inorder after deletion :\n");
    Display(Tree);
    printf("\n");
```

```
            break;
        case 6:
            printf("Tree elements in inorder :\n");
            Display(Tree);
            printf("\n");
            break;
        case 7:
            printf("Exiting...\n");
            break;
        default:
            printf("Invalid choice, please try again.\n");
      }
   } while (ch != 7);

   return 0;
}
```