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

typedef struct tree
{
    int data;
    struct tree *left;
    struct tree *right;
} node;

node *createnode(int element)
{
    node *ptr = (node *)malloc(sizeof(node));
    ptr->data = element;
    ptr->left = NULL;
    ptr->right = NULL;
    return (ptr);
}

void inorder(node *ptr)
{
    if (ptr == NULL)
    {
        return;
    }
    inorder(ptr->left);
    printf("%d ", ptr->data);
    inorder(ptr->right);
}

void postorder(node *ptr)
{
    if (ptr == NULL)
    {
        return;
    }
    postorder(ptr->left);
    postorder(ptr->right);
    printf("%d ", ptr->data);
}
void preorder(node *ptr)
{
    if (ptr == NULL)
    {
        return;
    }
    printf("%d ", ptr->data);
    preorder(ptr->left);
    preorder(ptr->right);
}

int getleveluntill(node *ptr, int data, int level)
{
    if (ptr == NULL)
    {
        return 0;
```

```
    }
    if (ptr->data == data)
    {
        return level;
    }
    int dlevel = getleveluntill(ptr->left, data, level + 1);
    if (dlevel != 0)
    {
        return dlevel;
    }
    dlevel = getleveluntill(ptr->right, data, level + 1);
    return dlevel;
}

int getlevel(node *ptr, int data)
{
    getleveluntill(ptr, data, 1);
}

void table(node *ptr)
{
    if (ptr->left == NULL)
    {
        return;
    }
    else
    {
        printf("Item\t Left Child\t Right Child\n");
        if (ptr->left != NULL && ptr->right != NULL)
        {
            printf("%d\t %d\t %d\t\n", ptr->data, ptr->left->data,
ptr->right->data);
        }
        else if (ptr->left != NULL)
        {
            printf("%d\t no child \t %d\t\n", ptr->data, ptr->right-
>data);
        }
        else
        {
            printf("%d\t%d\t no child\n", ptr->data, ptr->left->data);
        }
        table(ptr->left);
        table(ptr->right);
    }
}
node *insert(node *ptr, int key)
{
    if (ptr == NULL)
    {
        return createnode(key);
    }
    else if (key < ptr->data)
    {
        ptr->left = insert(ptr->left, key);
    }
    else
```

```
    {
        ptr->right = insert(ptr->right, key);
    }
    return (ptr);
}

node *minvalue(node *ptr)
{
    node *current = ptr;
    while (current && current->left != NULL)
    {
        current = current->left;
    }
    return current;
}

node *del(node *ptr, int key)
{
    if (ptr == NULL)
    {
        return ptr;
    }
    if (key < ptr->data)
    {
        ptr->left = del(ptr->left, key);
    }
    else if (key > ptr->data)
    {
        ptr->right = del(ptr->right, key);
    }
    else
    {
        if (ptr->left == NULL)
        {
            node *temp = ptr->right;
            free(ptr);
            return temp;
        }
        else if (ptr->right == NULL)
        {
            node *temp = ptr->left;
            free(ptr);
            return temp;
        }
        node *temp = minvalue(ptr->right);
        ptr->data = temp->data;
        ptr->right = del(ptr->right, temp->data);
    }
    return ptr;
}

int main()
{

    node *ptr = createnode(1);
    ptr->left = createnode(2);
    ptr->right = createnode(3);
```

```
    ptr->left->left = createnode(4);
    ptr->left->right = createnode(5);
    printf("The Preorder traversal is \n");
    preorder(ptr);
    printf("\n");
    printf("The Inorder Traversal is \n");
    inorder(ptr);
    printf("\n");
    printf("The Postorder Taversal is \n");
    postorder(ptr);
    printf("\n");

    int ele;
    printf("Enter the element you want to isnert\n");
    scanf("%d", &ele);
    insert(ptr, ele);
    printf("Level of %d is %d\n", ele, getlevel(ptr, ele));
    table(ptr);
    printf("Enter the element you want to insert\n");
    int ele1, ele2;
    scanf("%d", &ele1);
    printf(" The Inoder Traversal after the insertion of the element
is\n");
    inorder(ptr);
    printf("\nEnter the element you want to delete\n");
    scanf("%d", &ele2);
    del(ptr, ele2);
    printf("\n");
    printf("The Inorder Traversal after the deletion of the element
is\n ");
    inorder(ptr);

    return 0;
}
```