Assignment: 9

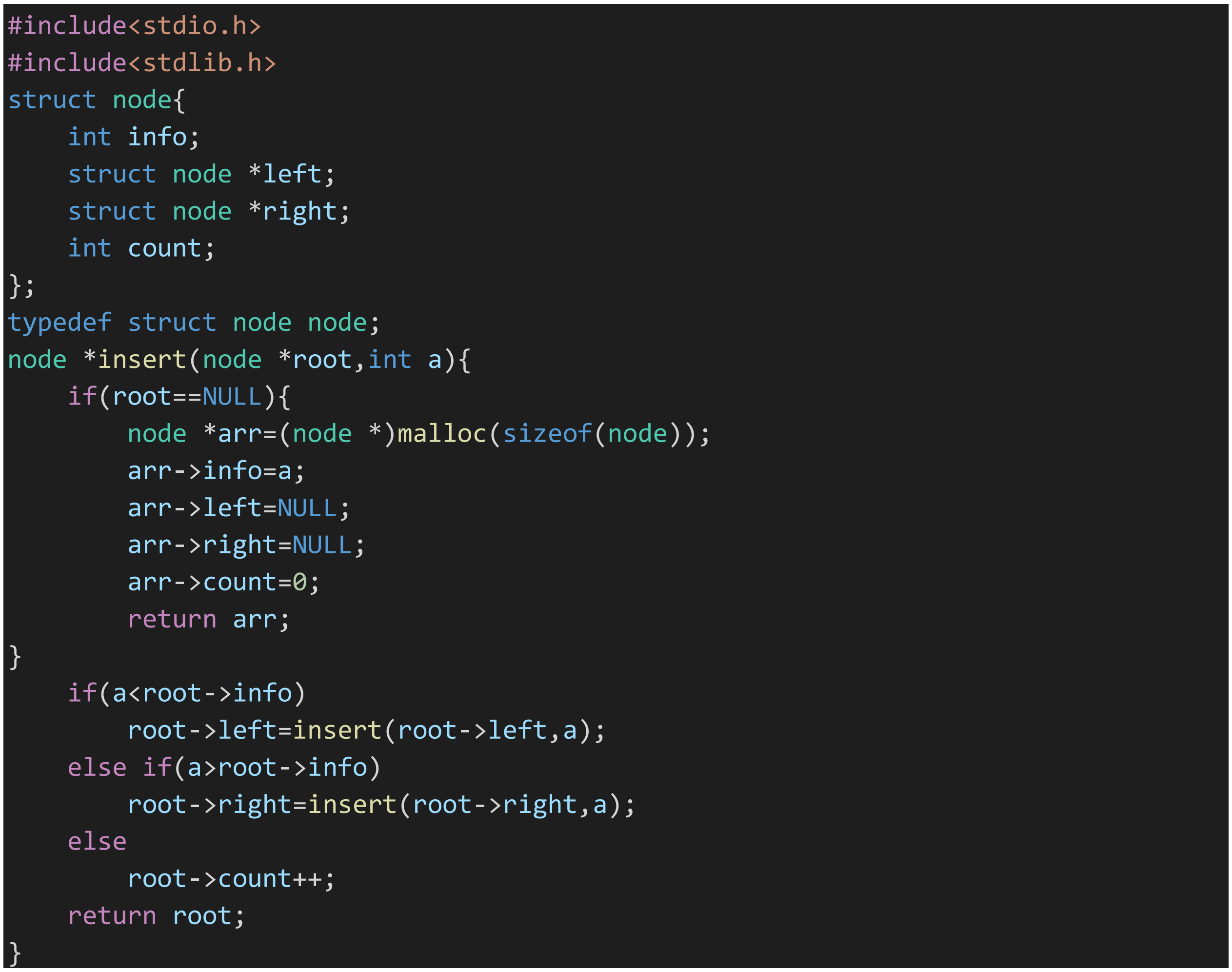
Name: Ujjwal Kumar Jha

Registration No. 20194196

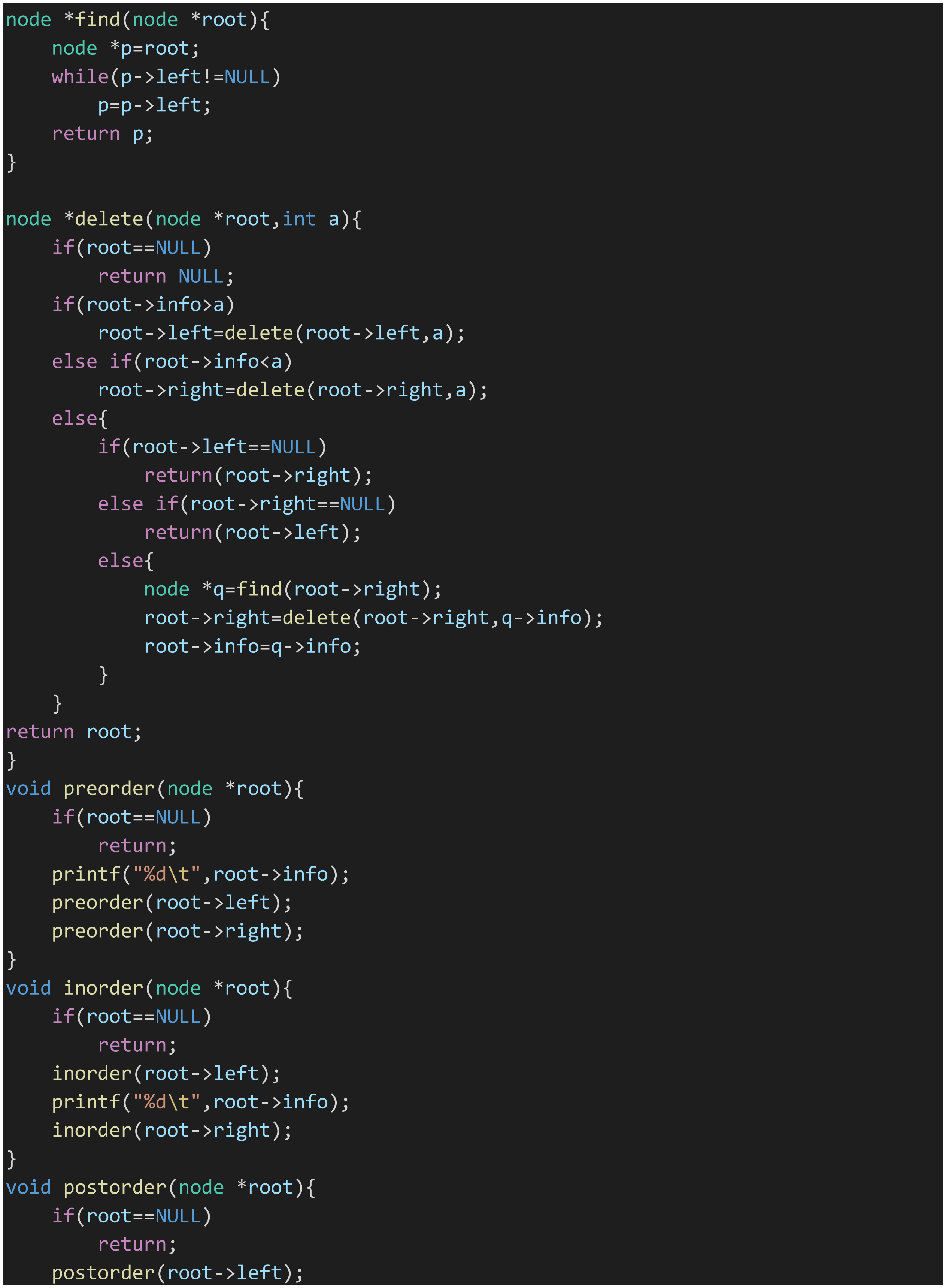
Group: C_2

Q1. WAP to implement Binary Search Tree (BST).

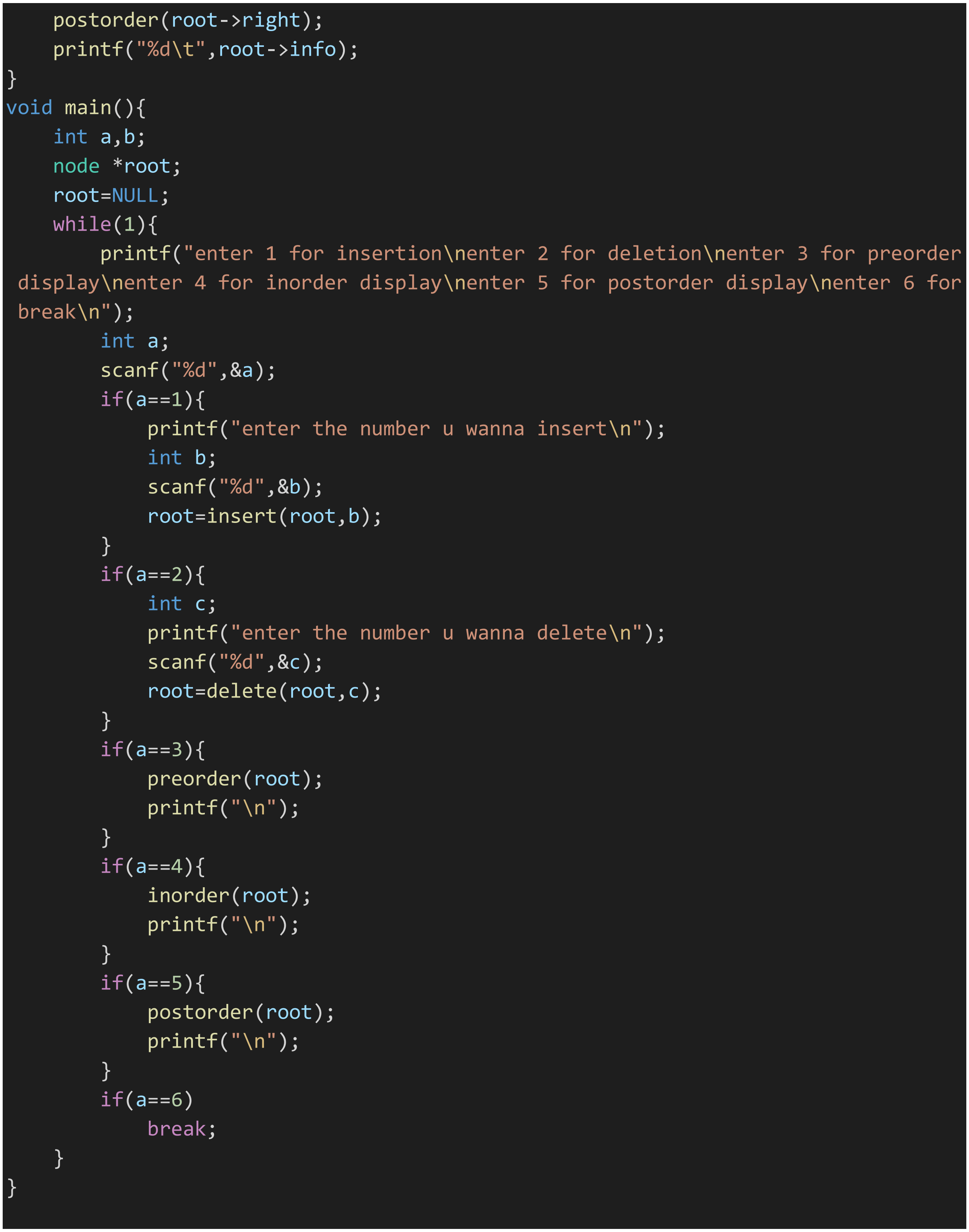
a. Insert a node into the BST.
b. Delete a node from the BST.
c. Display the preorder traversal of the BST.
d. Display the inorder traversal of the BST.
e. Display the postorder traversal of the BST.



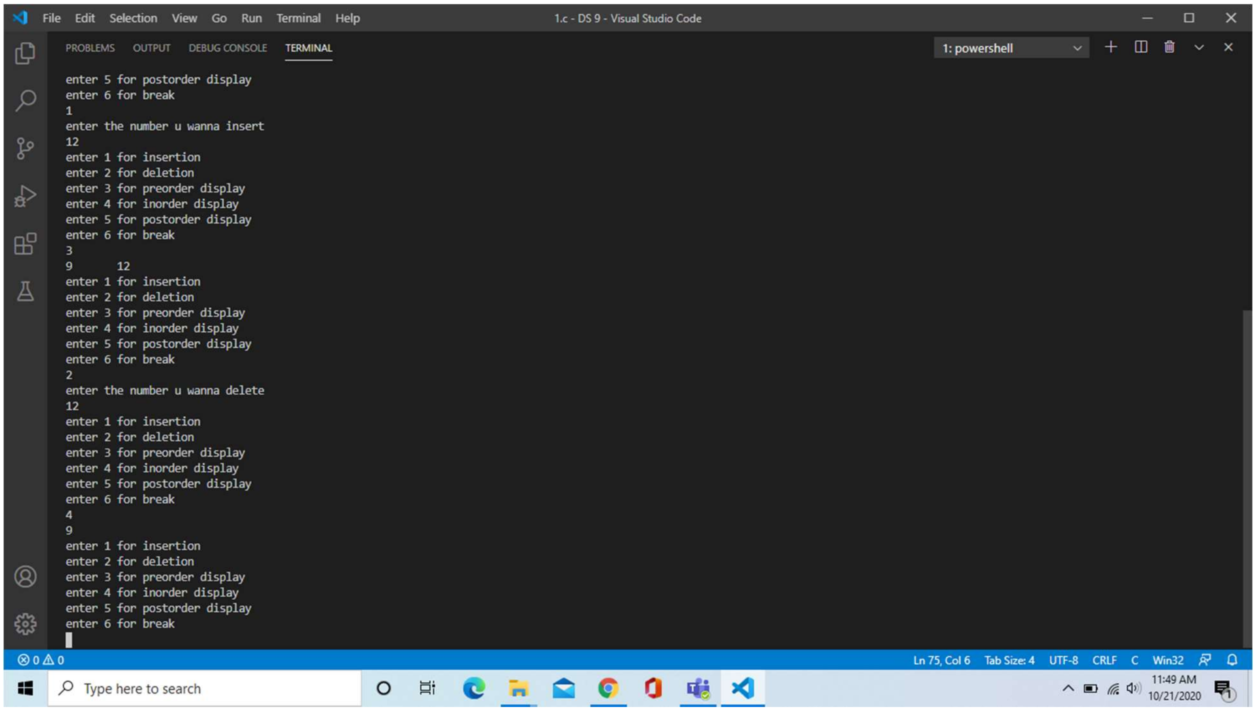
```c
#include<stdio.h>
#include<stdlib.h>
struct node{
    int info;
    struct node *left;
    struct node *right;
    int count;
};
typedef struct node node;
node *insert(node *root,int a){
    if(root==NULL){
        node *arr=(node *)malloc(sizeof(node));
        arr->info=a;
        arr->left=NULL;
        arr->right=NULL;
        arr->count=0;
        return arr;
}
    if(a<root->info)
        root->left=insert(root->left,a);
    else if(a>root->info)
        root->right=insert(root->right,a);
    else
        root->count++;
    return root;
}
```



```c
node *find(node *root){
    node *p=root;
    while(p->left!=NULL)
        p=p->left;
    return p;
}

node *delete(node *root,int a){
    if(root==NULL)
        return NULL;
    if(root->info>a)
        root->left=delete(root->left,a);
    else if(root->info<a)
        root->right=delete(root->right,a);
    else{
        if(root->left==NULL)
            return(root->right);
        else if(root->right==NULL)
            return(root->left);
        else{
            node *q=find(root->right);
            root->right=delete(root->right,q->info);
            root->info=q->info;
        }
    }
return root;
}
void preorder(node *root){
    if(root==NULL)
        return;
    printf("%d\t",root->info);
    preorder(root->left);
    preorder(root->right);
}
void inorder(node *root){
    if(root==NULL)
        return;
    inorder(root->left);
    printf("%d\t",root->info);
    inorder(root->right);
}
void postorder(node *root){
    if(root==NULL)
        return;
    postorder(root->left);
```



```
    postorder(root->right);
    printf("%d\t",root->info);
}
void main(){
    int a,b;
    node *root;
    root=NULL;
    while(1){
        printf("enter 1 for insertion\nenter 2 for deletion\nenter 3 for preorder
 display\nenter 4 for inorder display\nenter 5 for postorder display\nenter 6 for
 break\n");
        int a;
        scanf("%d",&a);
        if(a==1){
            printf("enter the number u wanna insert\n");
            int b;
            scanf("%d",&b);
            root=insert(root,b);
        }
        if(a==2){
            int c;
            printf("enter the number u wanna delete\n");
            scanf("%d",&c);
            root=delete(root,c);
        }
        if(a==3){
            preorder(root);
            printf("\n");
        }
        if(a==4){
            inorder(root);
            printf("\n");
        }
        if(a==5){
            postorder(root);
            printf("\n");
        }
        if(a==6)
            break;
    }
}
```



File Edit Selection View Go Run Terminal Help
1.c - DS 9 - Visual Studio Code
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: powershell
Ln 75, Col 6 Tab Size: 4 UTF-8 CRLF C Win32
Type here to search
11:49 AM
10/21/2020

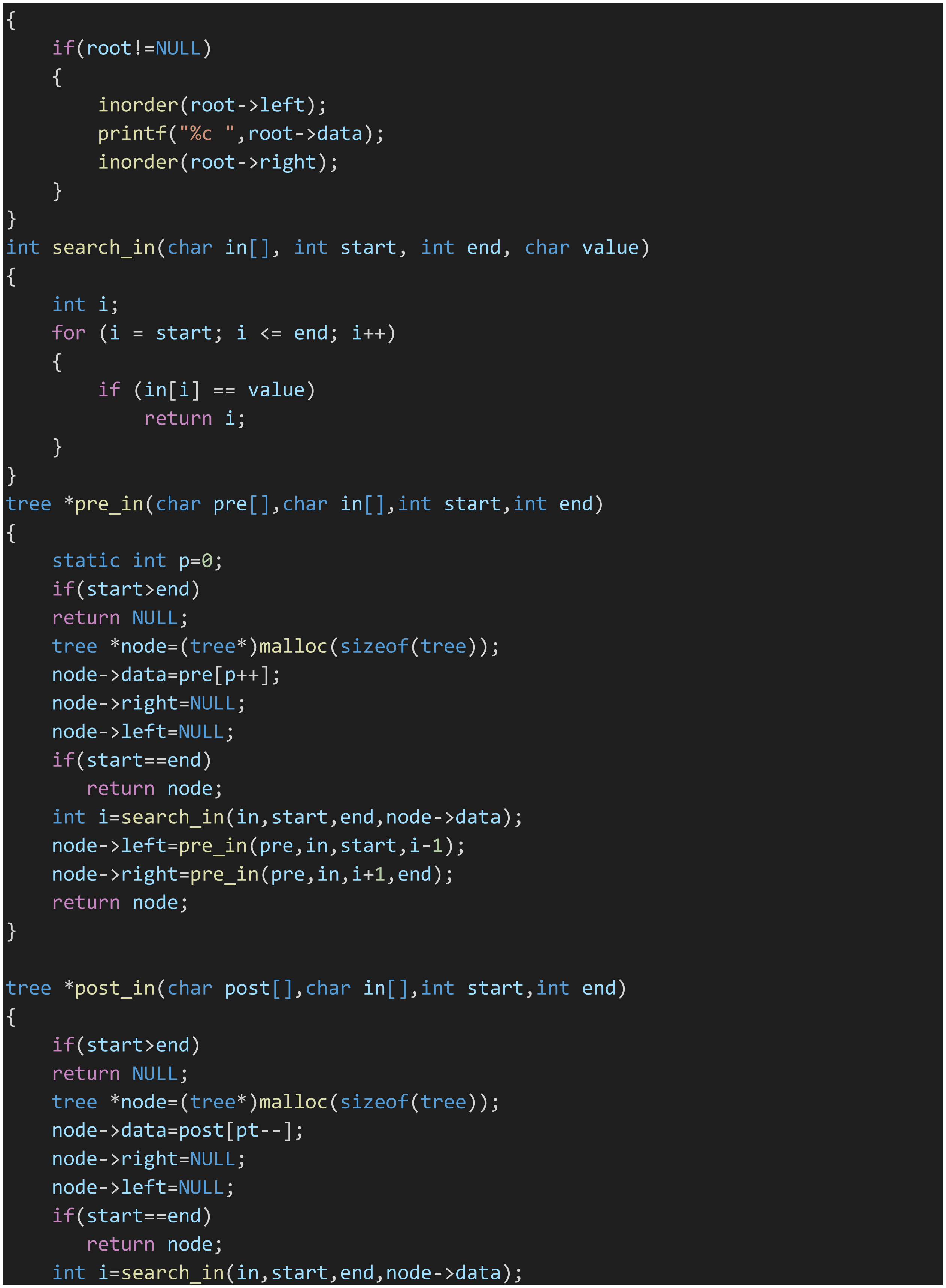
```
enter 5 for postorder display
enter 6 for break
1
enter the number u wanna insert
12
enter 1 for insertion
enter 2 for deletion
enter 3 for preorder display
enter 4 for inorder display
enter 5 for postorder display
enter 6 for break
3
9       12
enter 1 for insertion
enter 2 for deletion
enter 3 for preorder display
enter 4 for inorder display
enter 5 for postorder display
enter 6 for break
2
enter the number u wanna delete
12
enter 1 for insertion
enter 2 for deletion
enter 3 for preorder display
enter 4 for inorder display
enter 5 for postorder display
enter 6 for break
4
9
enter 1 for insertion
enter 2 for deletion
enter 3 for preorder display
enter 4 for inorder display
enter 5 for postorder display
enter 6 for break
```

2. WAP to construct a binary tree given

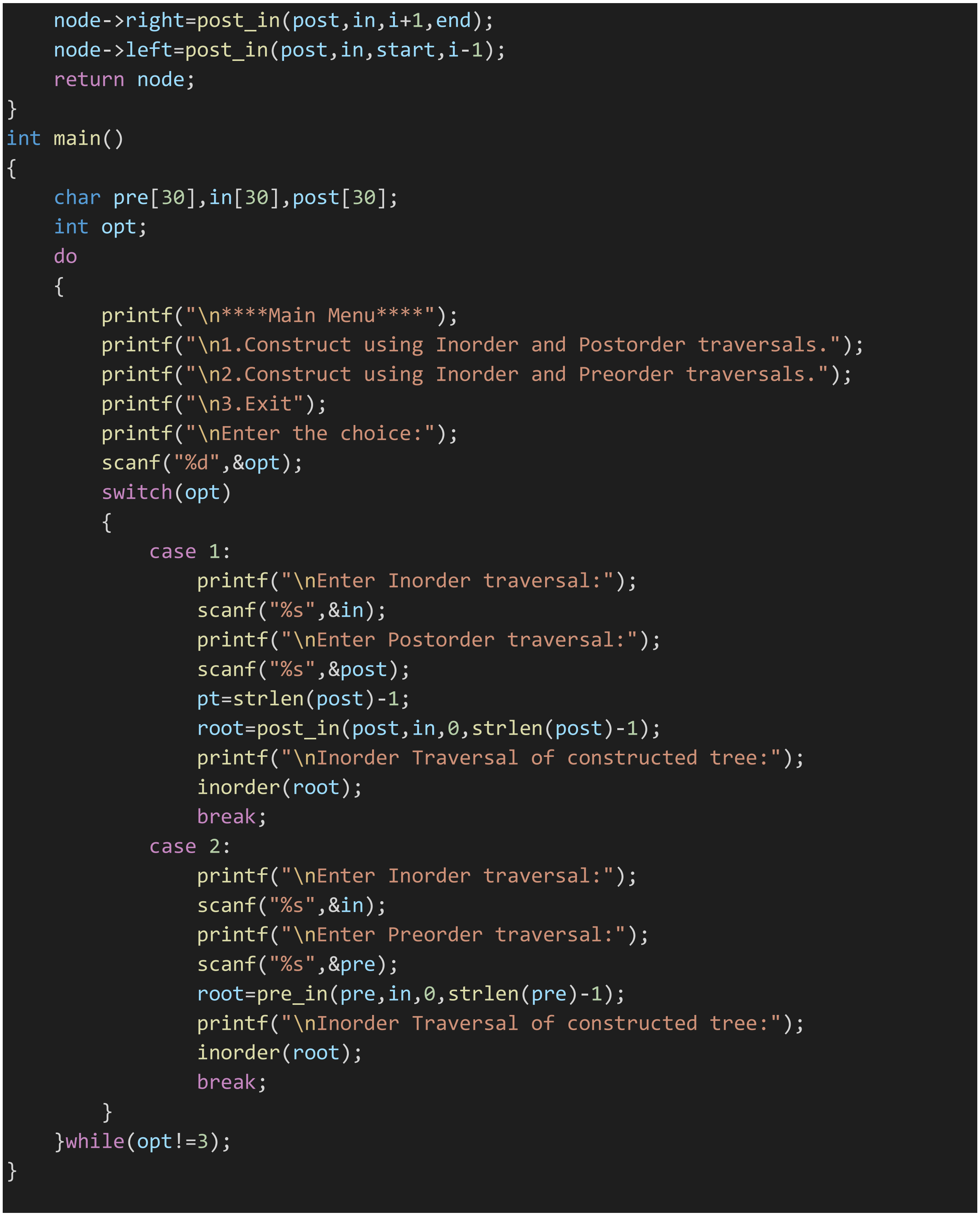
a. Inorder and Postorder traversals.

b. Inorder and Preorder traversals.

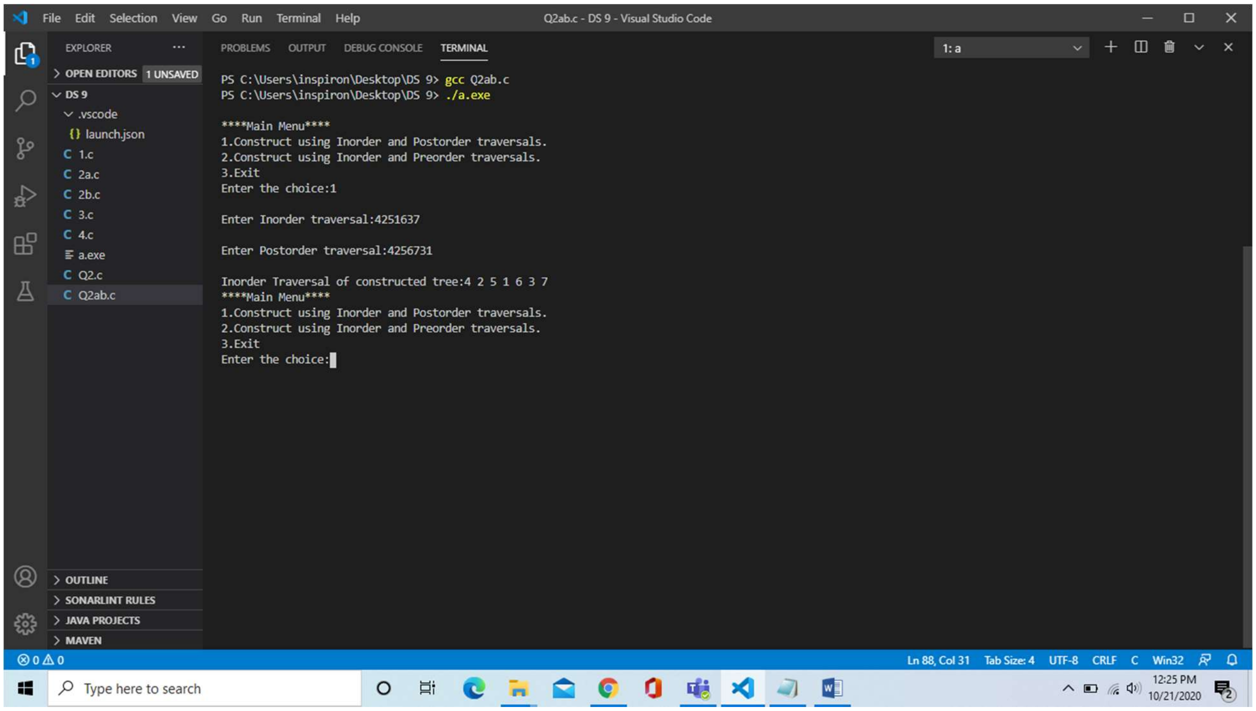
```
/* Robin Raj
20194033   */

#include <stdio.h>
#include <malloc.h>
#include <string.h>
#define tree struct node
int pt;
tree
{
    char data;
    tree *left;
    tree *right;
};
tree *root=NULL;
void inorder(tree *root)
```



```
{
    if(root!=NULL)
    {
        inorder(root->left);
        printf("%c ",root->data);
        inorder(root->right);
    }
}
int search_in(char in[], int start, int end, char value)
{
    int i;
    for (i = start; i <= end; i++)
    {
        if (in[i] == value)
            return i;
    }
}
tree *pre_in(char pre[],char in[],int start,int end)
{
    static int p=0;
    if(start>end)
    return NULL;
    tree *node=(tree*)malloc(sizeof(tree));
    node->data=pre[p++];
    node->right=NULL;
    node->left=NULL;
    if(start==end)
       return node;
    int i=search_in(in,start,end,node->data);
    node->left=pre_in(pre,in,start,i-1);
    node->right=pre_in(pre,in,i+1,end);
    return node;
}

tree *post_in(char post[],char in[],int start,int end)
{
    if(start>end)
    return NULL;
    tree *node=(tree*)malloc(sizeof(tree));
    node->data=post[pt--];
    node->right=NULL;
    node->left=NULL;
    if(start==end)
       return node;
    int i=search_in(in,start,end,node->data);
```



```
    node->right=post_in(post,in,i+1,end);
    node->left=post_in(post,in,start,i-1);
    return node;
}
int main()
{
    char pre[30],in[30],post[30];
    int opt;
    do
    {
        printf("\n****Main Menu****");
        printf("\n1.Construct using Inorder and Postorder traversals.");
        printf("\n2.Construct using Inorder and Preorder traversals.");
        printf("\n3.Exit");
        printf("\nEnter the choice:");
        scanf("%d",&opt);
        switch(opt)
        {
            case 1:
                printf("\nEnter Inorder traversal:");
                scanf("%s",&in);
                printf("\nEnter Postorder traversal:");
                scanf("%s",&post);
                pt=strlen(post)-1;
                root=post_in(post,in,0,strlen(post)-1);
                printf("\nInorder Traversal of constructed tree:");
                inorder(root);
                break;
            case 2:
                printf("\nEnter Inorder traversal:");
                scanf("%s",&in);
                printf("\nEnter Preorder traversal:");
                scanf("%s",&pre);
                root=pre_in(pre,in,0,strlen(pre)-1);
                printf("\nInorder Traversal of constructed tree:");
                inorder(root);
                break;
        }
    }while(opt!=3);
}
```



```
PS C:\Users\inspiron\Desktop\DS 9> gcc Q2ab.c
PS C:\Users\inspiron\Desktop\DS 9> ./a.exe

****Main Menu****
1.Construct using Inorder and Postorder traversals.
2.Construct using Inorder and Preorder traversals.
3.Exit
Enter the choice:1

Enter Inorder traversal:4251637

Enter Postorder traversal:4256731

Inorder Traversal of constructed tree:4 2 5 1 6 3 7
****Main Menu****
1.Construct using Inorder and Postorder traversals.
2.Construct using Inorder and Preorder traversals.
3.Exit
Enter the choice:
```

3. WAP to implement the following:

a. Count the number of nodes in a binary tree.

b. Count the number of leaf nodes in a binary tree.

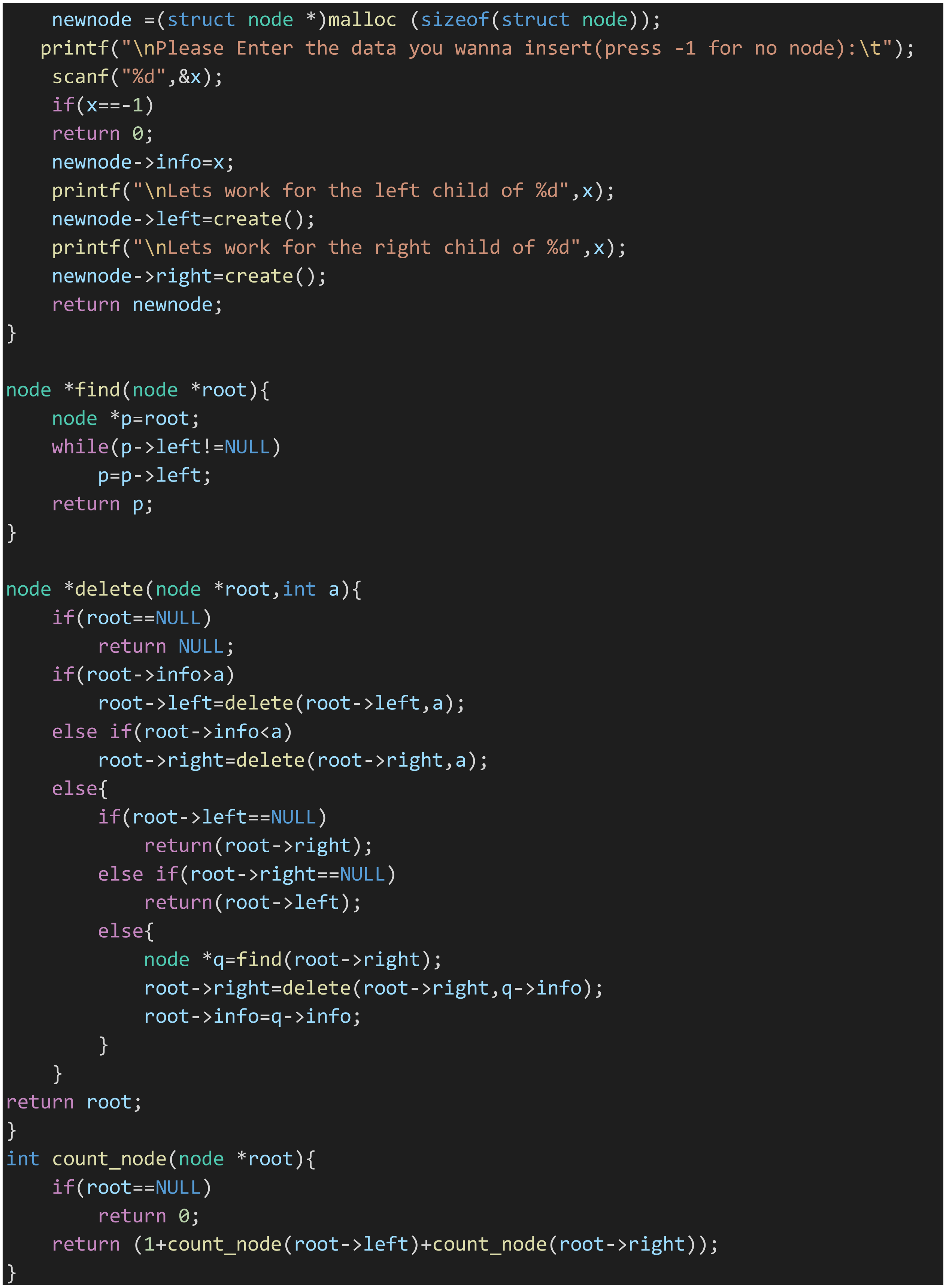
c. Count the number of non-leaf nodes in a binary tree.

d. Return the height of the binary tree.

e. Check whether the tree is a strict binary tree or not.

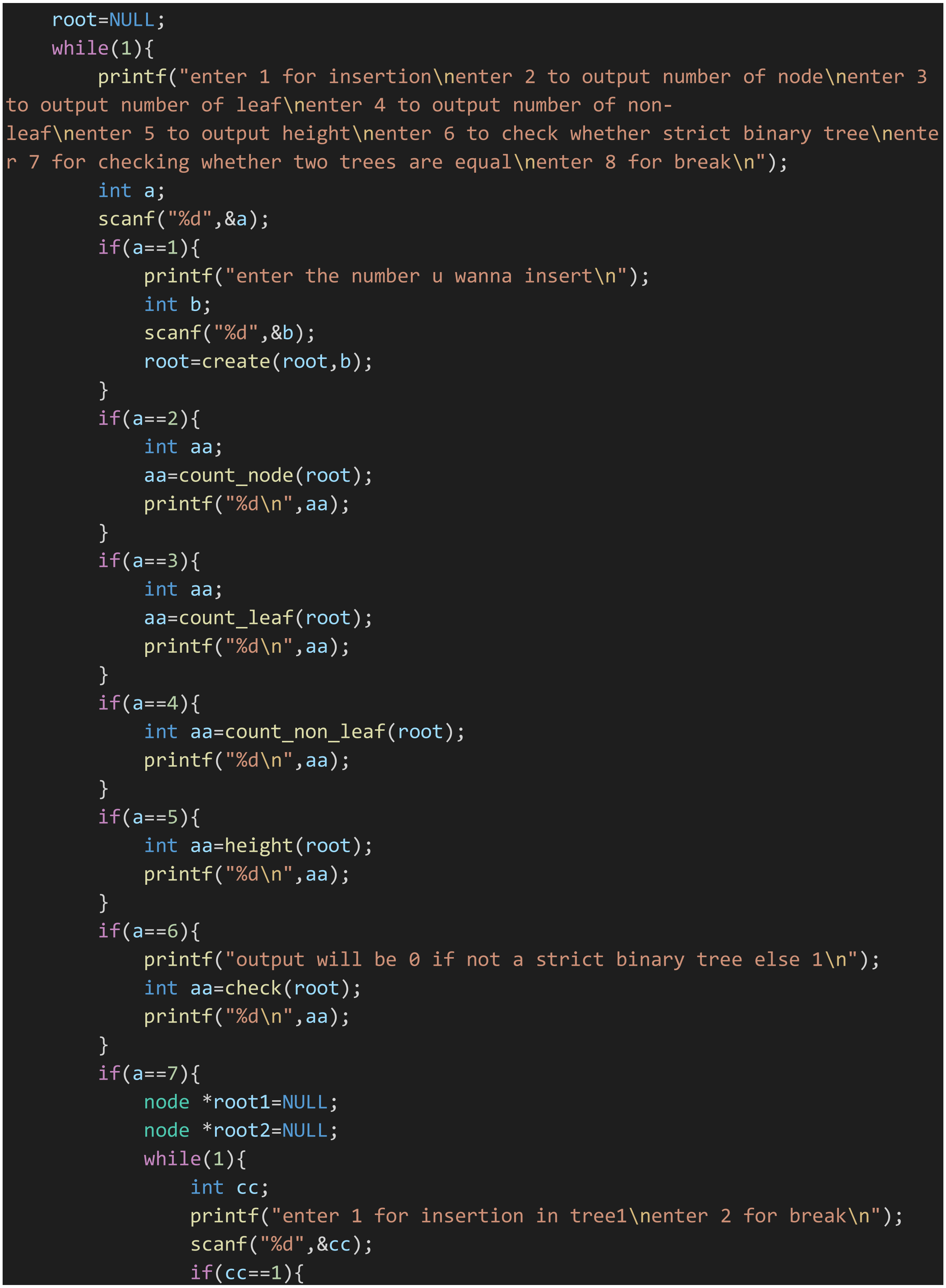
f. Check whether the two trees are equal or not.

```
#include<stdio.h>
#include<stdlib.h>
struct node{
    int info;
    struct node *left;
    struct node *right;
    int count;
};
typedef struct node node;
struct node *create(){
    int x;
    struct node *newnode;
```

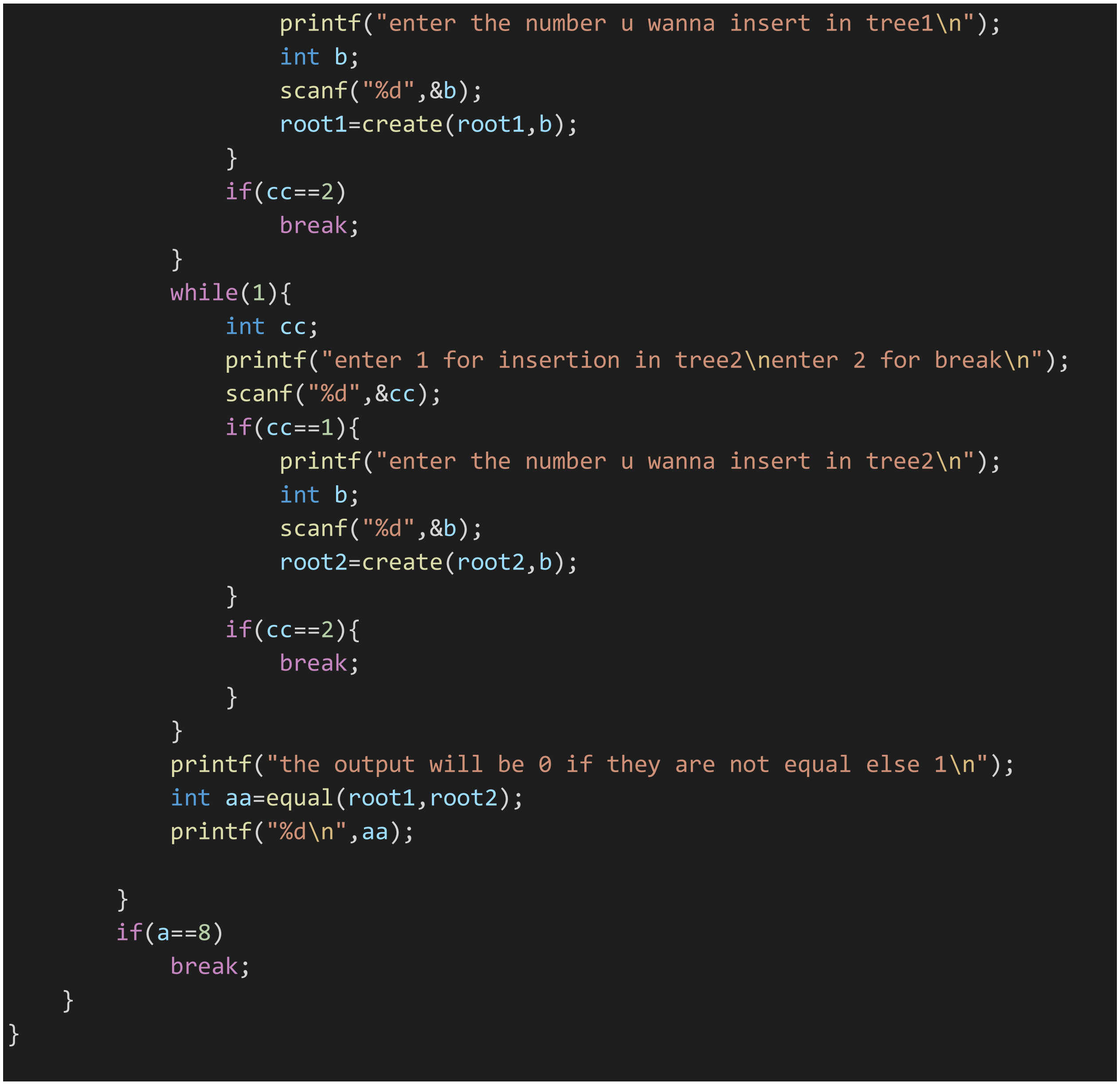


```
    newnode =(struct node *)malloc (sizeof(struct node));
   printf("\nPlease Enter the data you wanna insert(press -1 for no node):\t");
    scanf("%d",&x);
    if(x==-1)
    return 0;
    newnode->info=x;
    printf("\nLets work for the left child of %d",x);
    newnode->left=create();
    printf("\nLets work for the right child of %d",x);
    newnode->right=create();
    return newnode;
}

node *find(node *root){
    node *p=root;
    while(p->left!=NULL)
        p=p->left;
    return p;
}

node *delete(node *root,int a){
    if(root==NULL)
        return NULL;
    if(root->info>a)
        root->left=delete(root->left,a);
    else if(root->info<a)
        root->right=delete(root->right,a);
    else{
        if(root->left==NULL)
            return(root->right);
        else if(root->right==NULL)
            return(root->left);
        else{
            node *q=find(root->right);
            root->right=delete(root->right,q->info);
            root->info=q->info;
        }
    }
return root;
}
int count_node(node *root){
    if(root==NULL)
        return 0;
    return (1+count_node(root->left)+count_node(root->right));
}
```

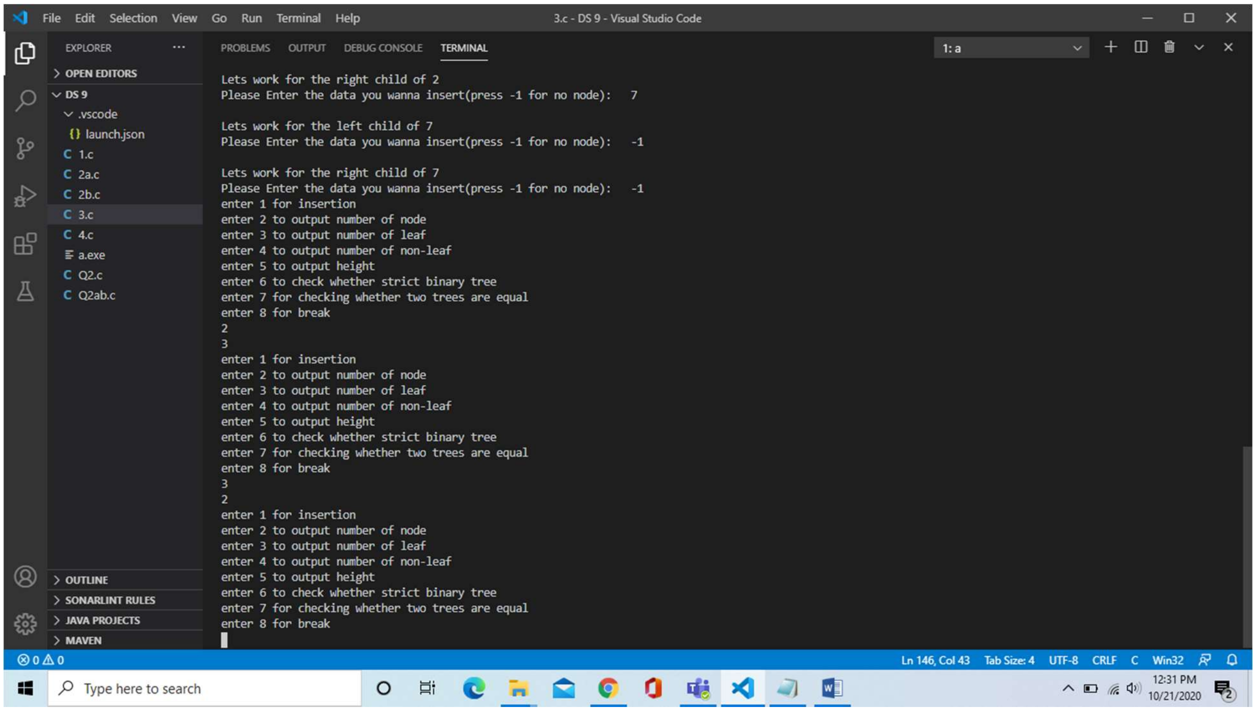
```
int count_leaf(node *root){
    if(root==NULL)
        return 0;
    if(root->left==NULL && root->right==NULL)
        return 1;
    return count_leaf(root->left)+count_leaf(root->right);
}
int count_non_leaf(node *root){
    if (root==NULL)
        return 0;
    if(root->left==NULL && root->right==NULL)
        return 0;
    return 1+count_non_leaf(root->left)+count_non_leaf(root->right);
}
int max(int a,int b){
    if(a>b)
        return a;
    else
        return b;
}
int height(node *root){
    if(root==NULL)
        return 0;
    return 1+max(height(root->left),height(root->right));
}
int check(node *root){
    if(root==NULL)
        return 1;
    if((root->left==NULL && root->right!=NULL) || (root->right==NULL && root-
>left!=NULL))
        return 0;
    return (check(root->left)&& check(root->right));
}
int equal(node *root1,node *root2){
    if(root1==NULL && root2==NULL)
        return 1;
    if((root1==NULL && root2!=NULL) || (root2==NULL && root1!=NULL)|| (root1-
>info!=root2->info))
        return 0;
    return (equal(root1->left,root2->left) && equal(root1->right,root2->right));
}

void main(){
    int a,b;
    node *root;
```



```c
    root=NULL;
    while(1){
        printf("enter 1 for insertion\nenter 2 to output number of node\nenter 3 to output number of leaf\nenter 4 to output number of non-leaf\nenter 5 to output height\nenter 6 to check whether strict binary tree\nenter 7 for checking whether two trees are equal\nenter 8 for break\n");
        int a;
        scanf("%d",&a);
        if(a==1){
            printf("enter the number u wanna insert\n");
            int b;
            scanf("%d",&b);
            root=create(root,b);
        }
        if(a==2){
            int aa;
            aa=count_node(root);
            printf("%d\n",aa);
        }
        if(a==3){
            int aa;
            aa=count_leaf(root);
            printf("%d\n",aa);
        }
        if(a==4){
            int aa=count_non_leaf(root);
            printf("%d\n",aa);
        }
        if(a==5){
            int aa=height(root);
            printf("%d\n",aa);
        }
        if(a==6){
            printf("output will be 0 if not a strict binary tree else 1\n");
            int aa=check(root);
            printf("%d\n",aa);
        }
        if(a==7){
            node *root1=NULL;
            node *root2=NULL;
            while(1){
                int cc;
                printf("enter 1 for insertion in tree1\nenter 2 for break\n");
                scanf("%d",&cc);
                if(cc==1){
```



```
                    printf("enter the number u wanna insert in tree1\n");
                    int b;
                    scanf("%d",&b);
                    root1=create(root1,b);
                }
                if(cc==2)
                    break;
            }
            while(1){
                int cc;
                printf("enter 1 for insertion in tree2\nenter 2 for break\n");
                scanf("%d",&cc);
                if(cc==1){
                    printf("enter the number u wanna insert in tree2\n");
                    int b;
                    scanf("%d",&b);
                    root2=create(root2,b);
                }
                if(cc==2){
                    break;
                }
            }
            printf("the output will be 0 if they are not equal else 1\n");
            int aa=equal(root1,root2);
            printf("%d\n",aa);

        }
        if(a==8)
            break;
    }
}
```


File Edit Selection View Go Run Terminal Help
3.c - DS 9 - Visual Studio Code
EXPLORER
OPEN EDITORS
DS 9
.vscode
launch.json
1.c
2a.c
2b.c
3.c
4.c
a.exe
Q2.c
Q2ab.c
OUTLINE
SONARLINT RULES
JAVA PROJECTS
MAVEN
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
1: a
Ln 146, Col 43 Tab Size: 4 UTF-8 CRLF C Win32
Type here to search
12:31 PM
10/21/2020

```
Lets work for the right child of 2
Please Enter the data you wanna insert(press -1 for no node):   7

Lets work for the left child of 7
Please Enter the data you wanna insert(press -1 for no node):   -1

Lets work for the right child of 7
Please Enter the data you wanna insert(press -1 for no node):   -1
enter 1 for insertion
enter 2 to output number of node
enter 3 to output number of leaf
enter 4 to output number of non-leaf
enter 5 to output height
enter 6 to check whether strict binary tree
enter 7 for checking whether two trees are equal
enter 8 for break
2
3
enter 1 for insertion
enter 2 to output number of node
enter 3 to output number of leaf
enter 4 to output number of non-leaf
enter 5 to output height
enter 6 to check whether strict binary tree
enter 7 for checking whether two trees are equal
enter 8 for break
3
2
enter 1 for insertion
enter 2 to output number of node
enter 3 to output number of leaf
enter 4 to output number of non-leaf
enter 5 to output height
enter 6 to check whether strict binary tree
enter 7 for checking whether two trees are equal
enter 8 for break
```

4. WAP to implement Threaded Binary Tree (TBT).

a. Insert a node into the TBT.

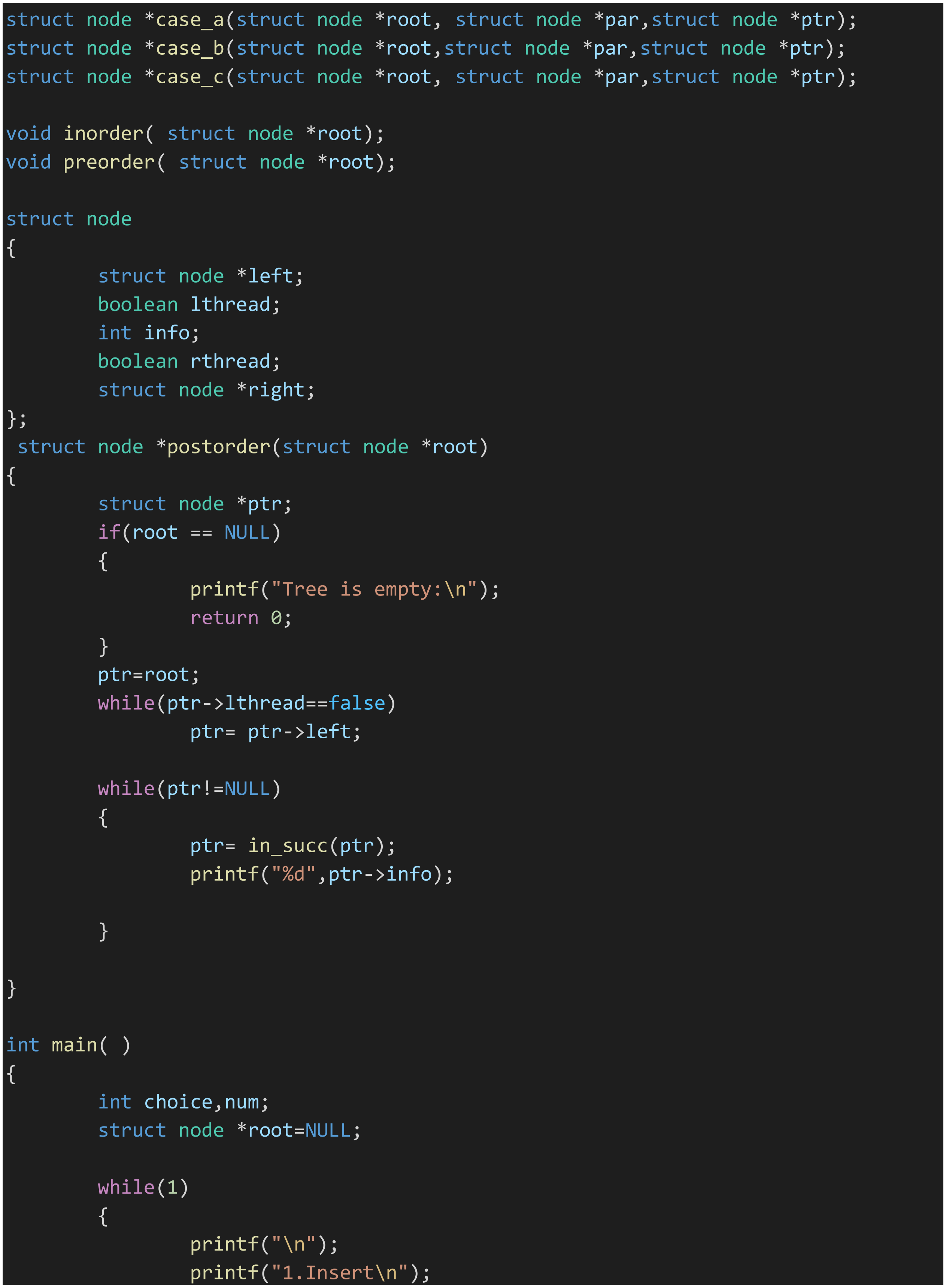
b. Delete a node from the TBT.

c. Display the preorder traversal of the TBT.

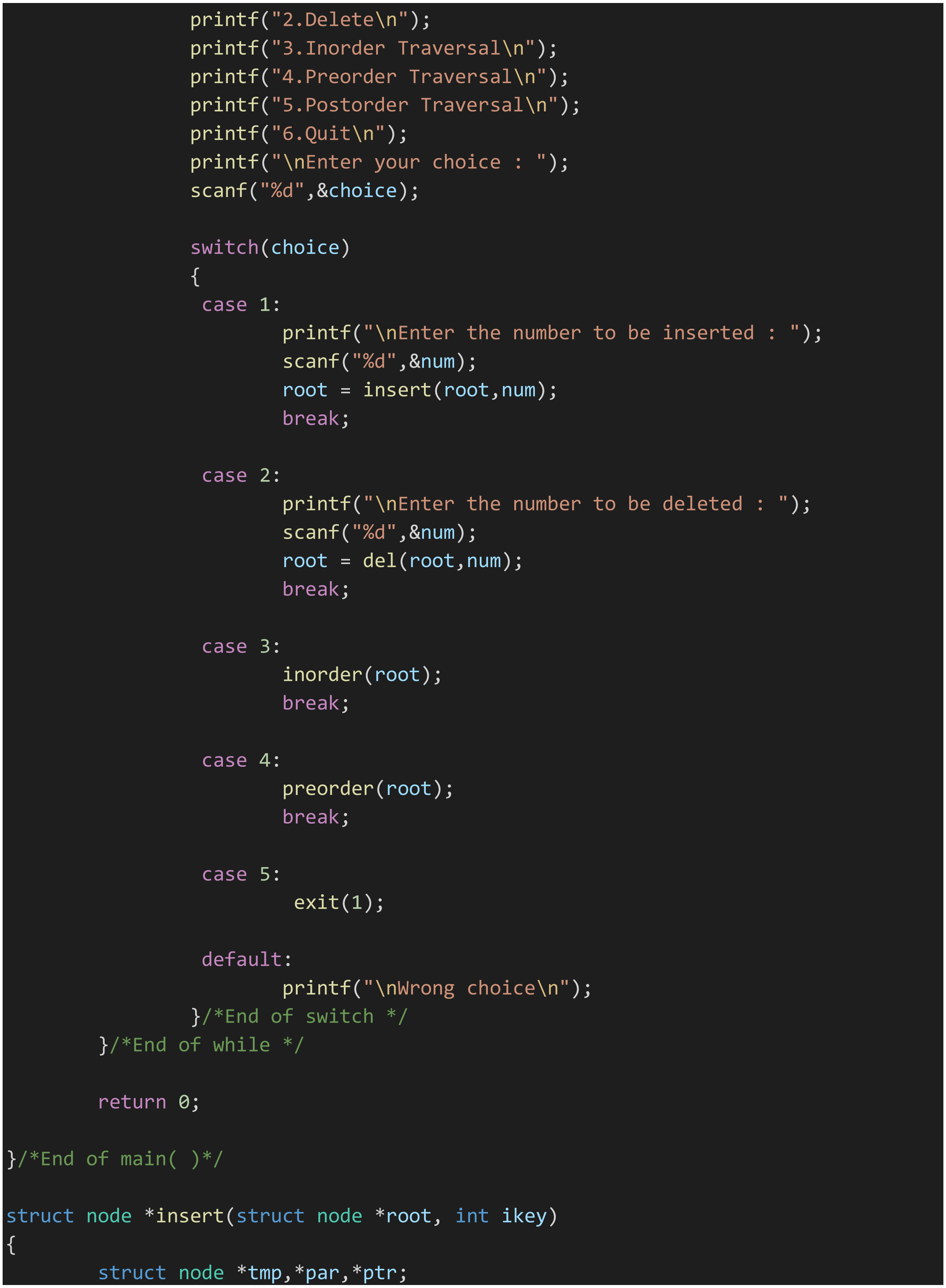
d. Display the inorder traversal of the TBT.

e. Display the postorder traversal of the TBT.

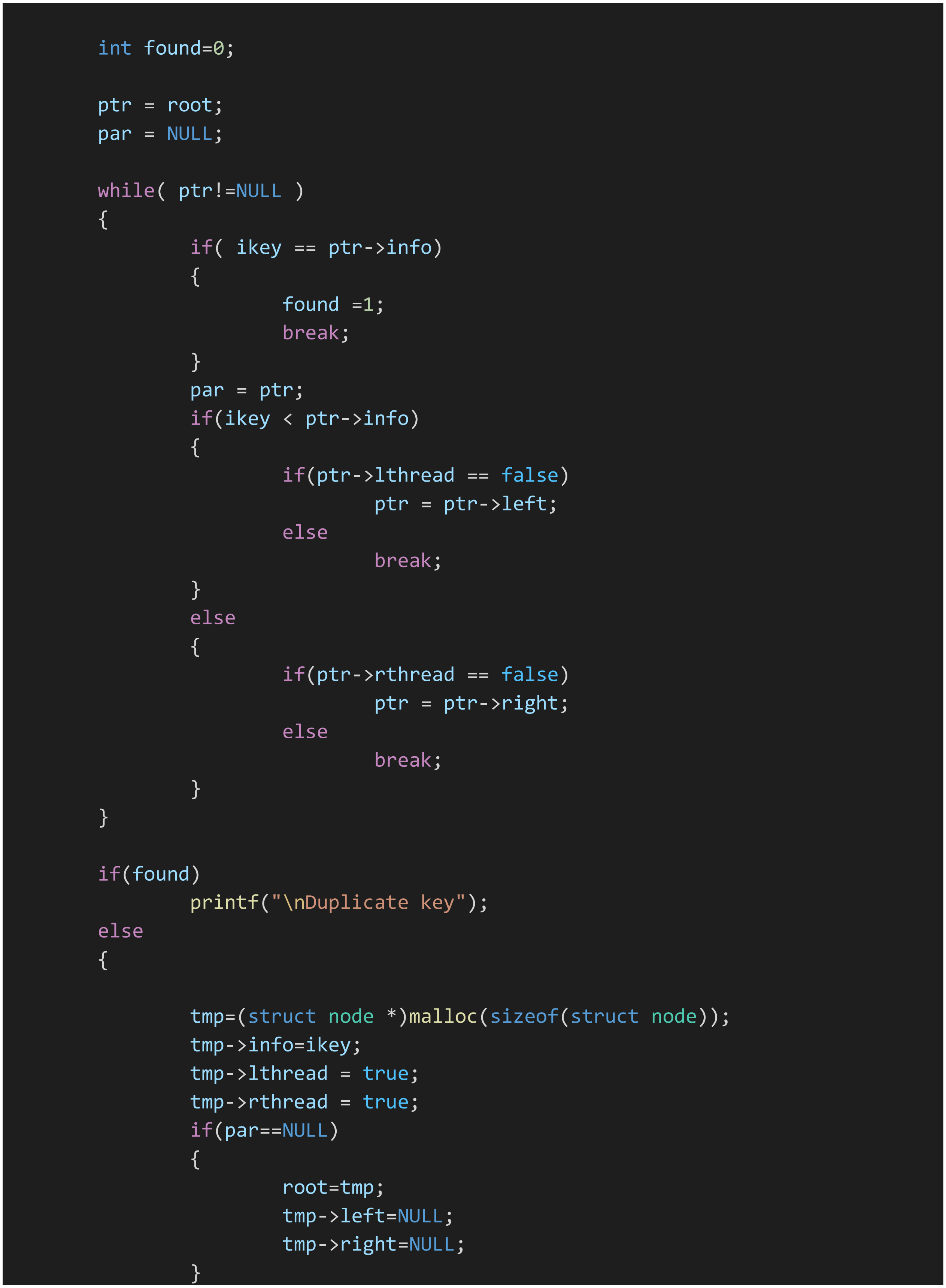
```c
#include <stdio.h>
#include <stdlib.h>

typedef enum {false,true} boolean;

struct node *in_succ(struct node *p);
struct node *in_pred(struct node *p);
struct node *insert(struct node *root, int ikey);
struct node *del(struct node *root, int dkey);
```



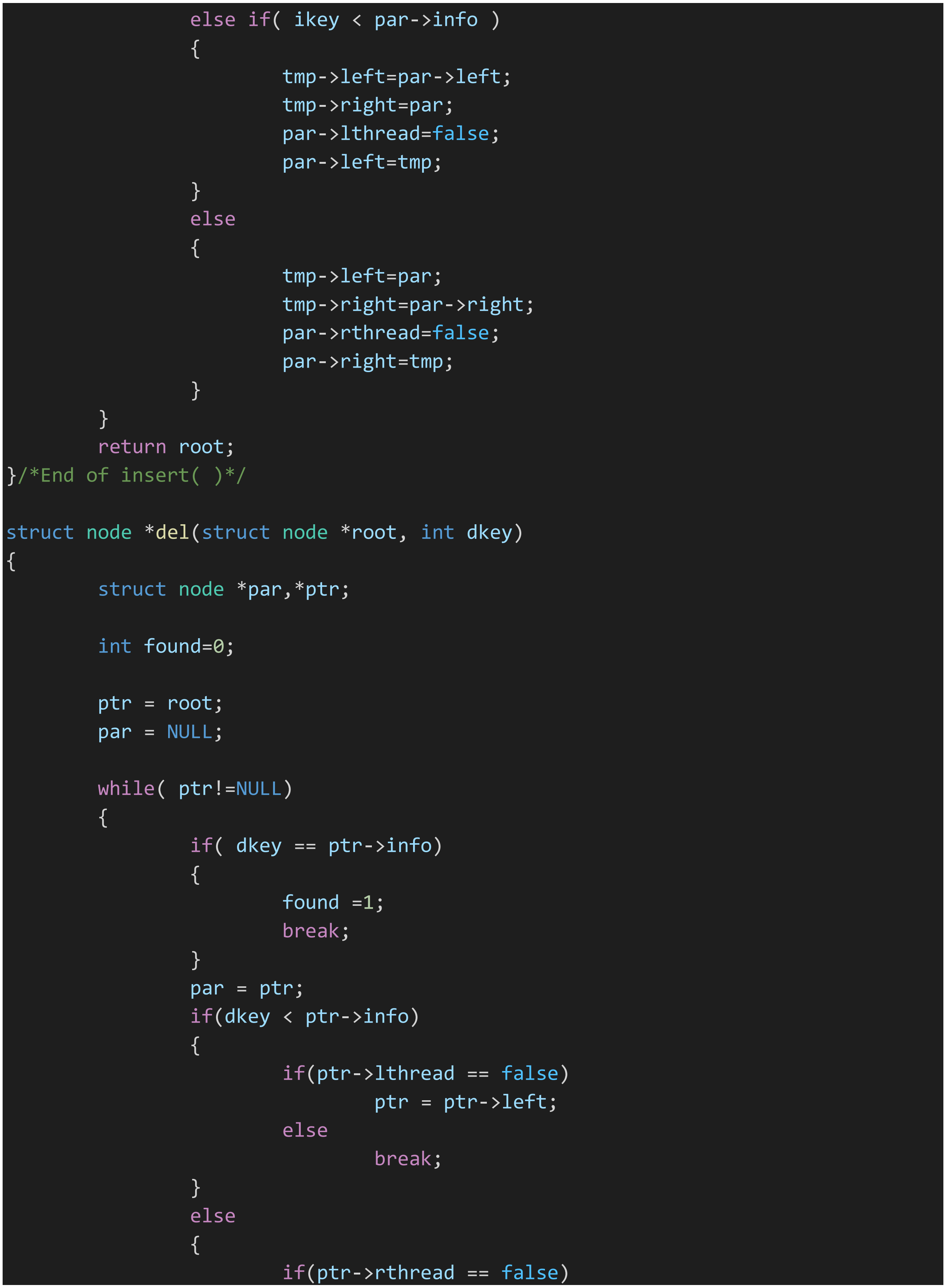
```
struct node *case_a(struct node *root, struct node *par,struct node *ptr);
struct node *case_b(struct node *root,struct node *par,struct node *ptr);
struct node *case_c(struct node *root, struct node *par,struct node *ptr);

void inorder( struct node *root);
void preorder( struct node *root);

struct node
{
        struct node *left;
        boolean lthread;
        int info;
        boolean rthread;
        struct node *right;
};
 struct node *postorder(struct node *root)
{
        struct node *ptr;
        if(root == NULL)
        {
                printf("Tree is empty:\n");
                return 0;
        }
        ptr=root;
        while(ptr->lthread==false)
                ptr= ptr->left;

        while(ptr!=NULL)
        {
                ptr= in_succ(ptr);
                printf("%d",ptr->info);

        }

}

int main( )
{
        int choice,num;
        struct node *root=NULL;

        while(1)
        {
                printf("\n");
                printf("1.Insert\n");
```



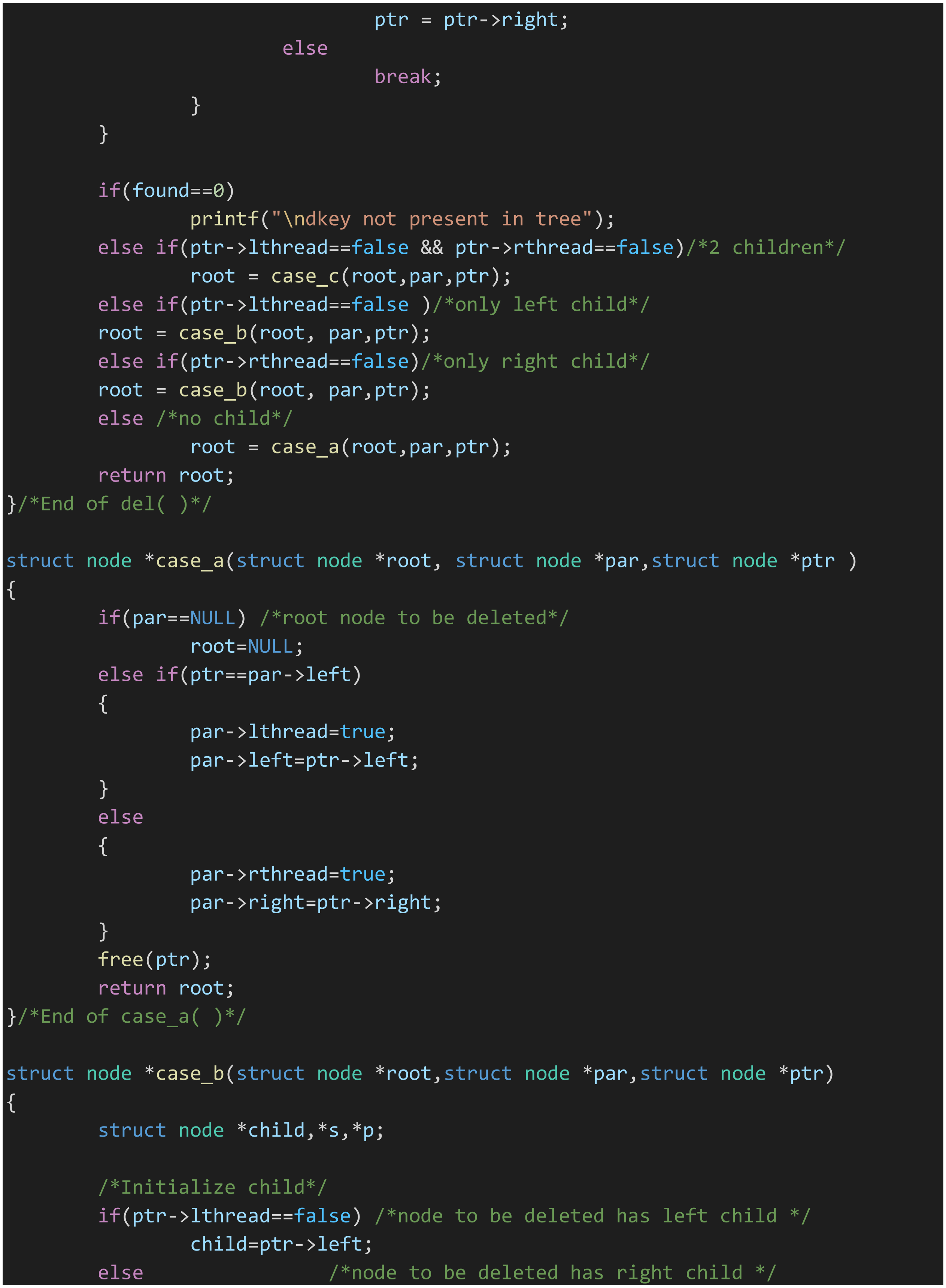
```
                printf("2.Delete\n");
                printf("3.Inorder Traversal\n");
                printf("4.Preorder Traversal\n");
                printf("5.Postorder Traversal\n");
                printf("6.Quit\n");
                printf("\nEnter your choice : ");
                scanf("%d",&choice);

                switch(choice)
                {
                 case 1:
                        printf("\nEnter the number to be inserted : ");
                        scanf("%d",&num);
                        root = insert(root,num);
                        break;

                 case 2:
                        printf("\nEnter the number to be deleted : ");
                        scanf("%d",&num);
                        root = del(root,num);
                        break;

                 case 3:
                        inorder(root);
                        break;

                 case 4:
                        preorder(root);
                        break;

                 case 5:
                         exit(1);

                 default:
                        printf("\nWrong choice\n");
                }/*End of switch */
        }/*End of while */

        return 0;

}/*End of main( )*/

struct node *insert(struct node *root, int ikey)
{
        struct node *tmp,*par,*ptr;
```



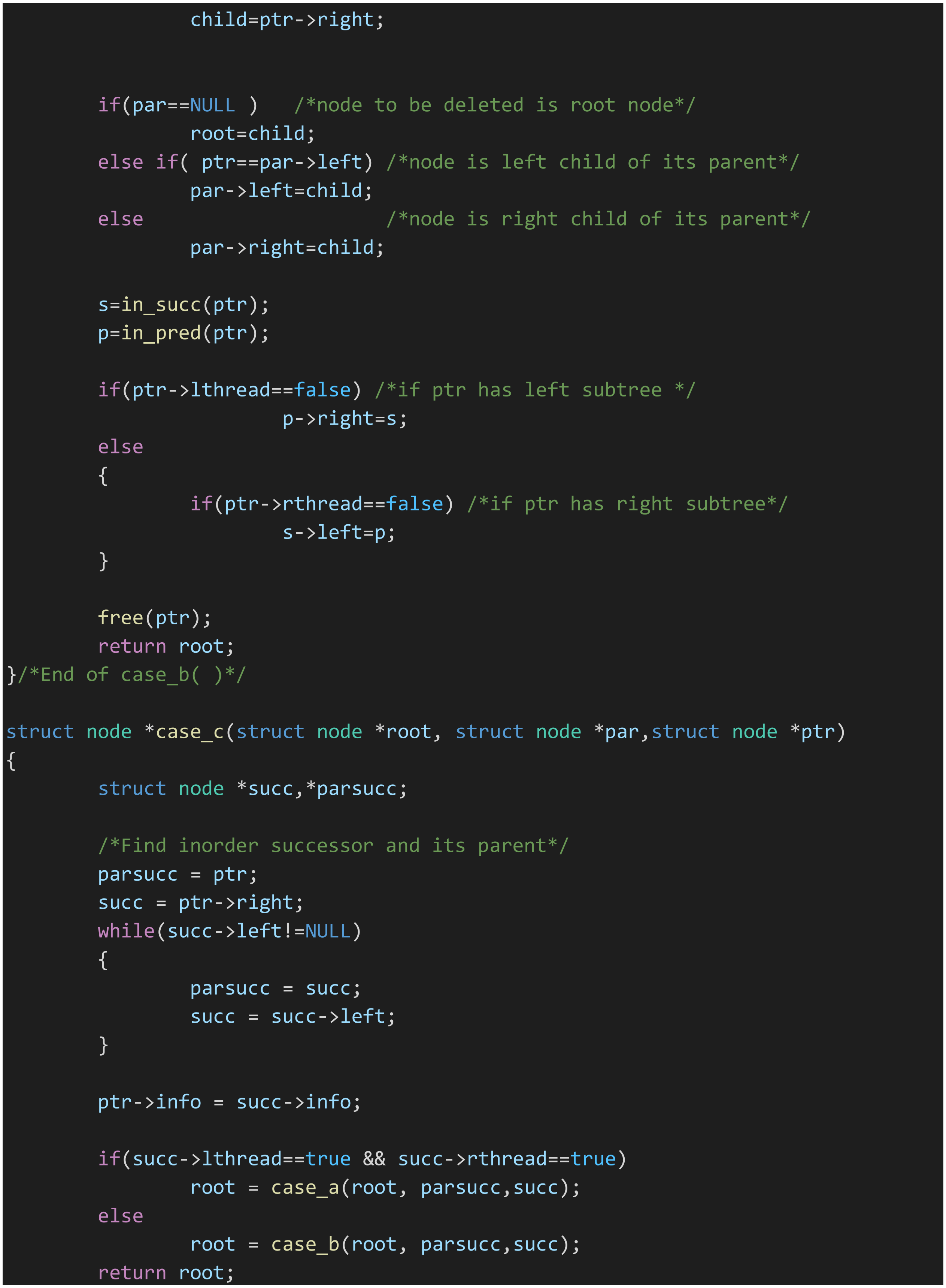
```
        int found=0;

        ptr = root;
        par = NULL;

        while( ptr!=NULL )
        {
                if( ikey == ptr->info)
                {
                        found =1;
                        break;
                }
                par = ptr;
                if(ikey < ptr->info)
                {
                        if(ptr->lthread == false)
                                ptr = ptr->left;
                        else
                                break;
                }
                else
                {
                        if(ptr->rthread == false)
                                ptr = ptr->right;
                        else
                                break;
                }
        }

        if(found)
                printf("\nDuplicate key");
        else
        {

                tmp=(struct node *)malloc(sizeof(struct node));
                tmp->info=ikey;
                tmp->lthread = true;
                tmp->rthread = true;
                if(par==NULL)
                {
                        root=tmp;
                        tmp->left=NULL;
                        tmp->right=NULL;
                }
```



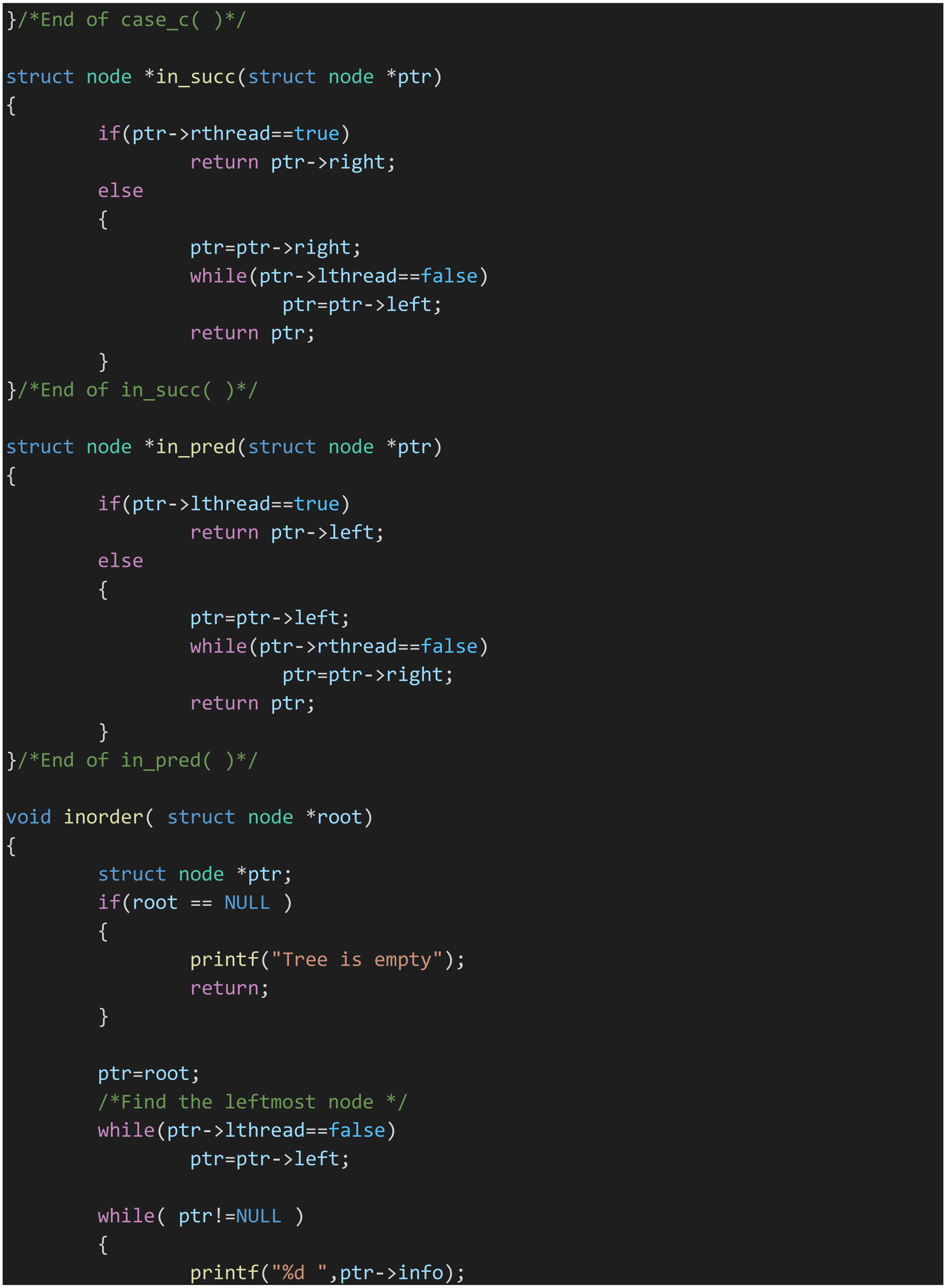
```
                else if( ikey < par->info )
                {
                        tmp->left=par->left;
                        tmp->right=par;
                        par->lthread=false;
                        par->left=tmp;
                }
                else
                {
                        tmp->left=par;
                        tmp->right=par->right;
                        par->rthread=false;
                        par->right=tmp;
                }
        }
        return root;
}/*End of insert( )*/

struct node *del(struct node *root, int dkey)
{
        struct node *par,*ptr;

        int found=0;

        ptr = root;
        par = NULL;

        while( ptr!=NULL)
        {
                if( dkey == ptr->info)
                {
                        found =1;
                        break;
                }
                par = ptr;
                if(dkey < ptr->info)
                {
                        if(ptr->lthread == false)
                                ptr = ptr->left;
                        else
                                break;
                }
                else
                {
                        if(ptr->rthread == false)
```



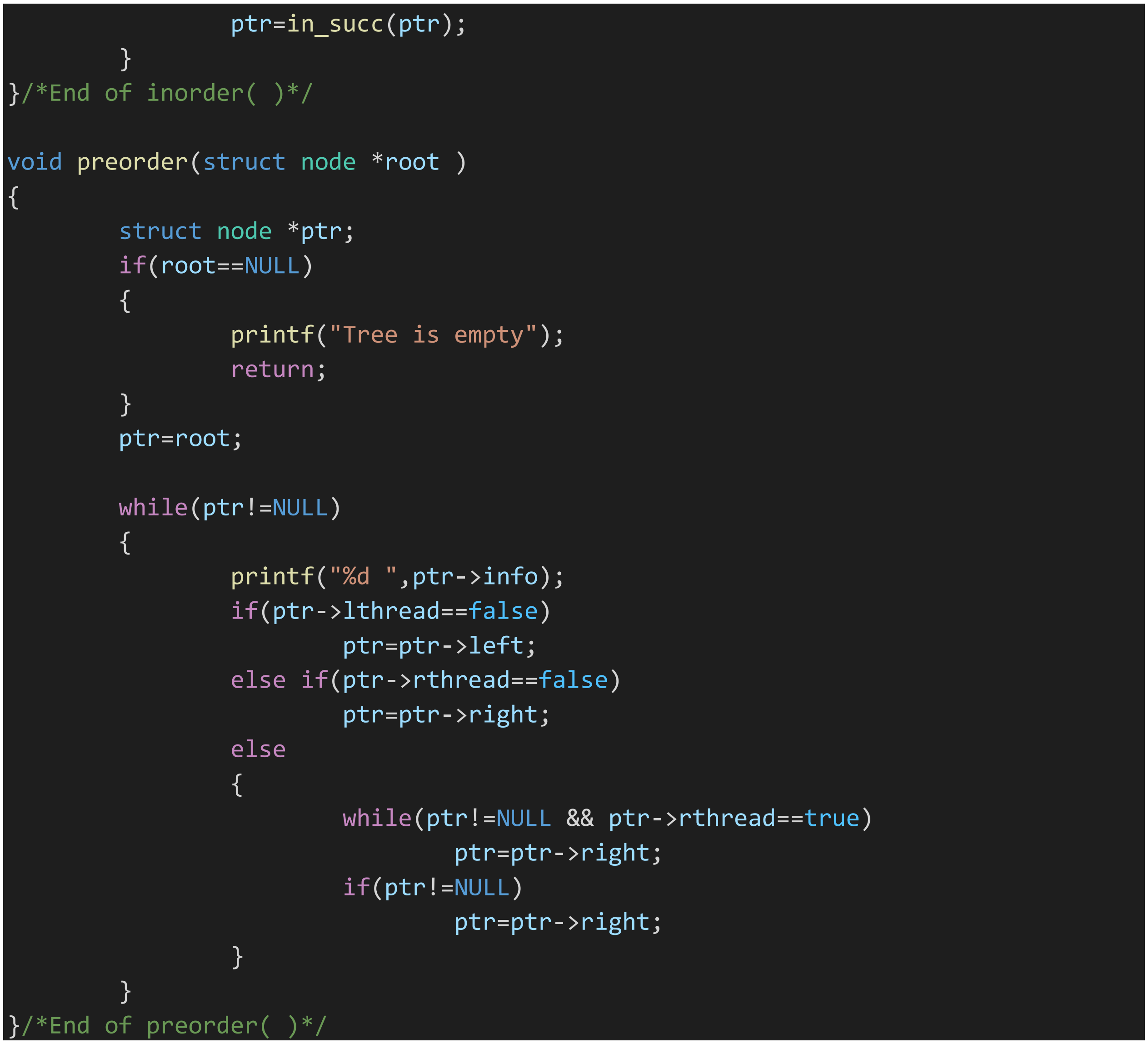
```c
                                ptr = ptr->right;
                        else
                                break;
                }
        }

        if(found==0)
                printf("\ndkey not present in tree");
        else if(ptr->lthread==false && ptr->rthread==false)/*2 children*/
                root = case_c(root,par,ptr);
        else if(ptr->lthread==false )/*only left child*/
        root = case_b(root, par,ptr);
        else if(ptr->rthread==false)/*only right child*/
        root = case_b(root, par,ptr);
        else /*no child*/
                root = case_a(root,par,ptr);
        return root;
}/*End of del( )*/

struct node *case_a(struct node *root, struct node *par,struct node *ptr )
{
        if(par==NULL) /*root node to be deleted*/
                root=NULL;
        else if(ptr==par->left)
        {
                par->lthread=true;
                par->left=ptr->left;
        }
        else
        {
                par->rthread=true;
                par->right=ptr->right;
        }
        free(ptr);
        return root;
}/*End of case_a( )*/

struct node *case_b(struct node *root,struct node *par,struct node *ptr)
{
        struct node *child,*s,*p;

        /*Initialize child*/
        if(ptr->lthread==false) /*node to be deleted has left child */
                child=ptr->left;
        else                /*node to be deleted has right child */
```



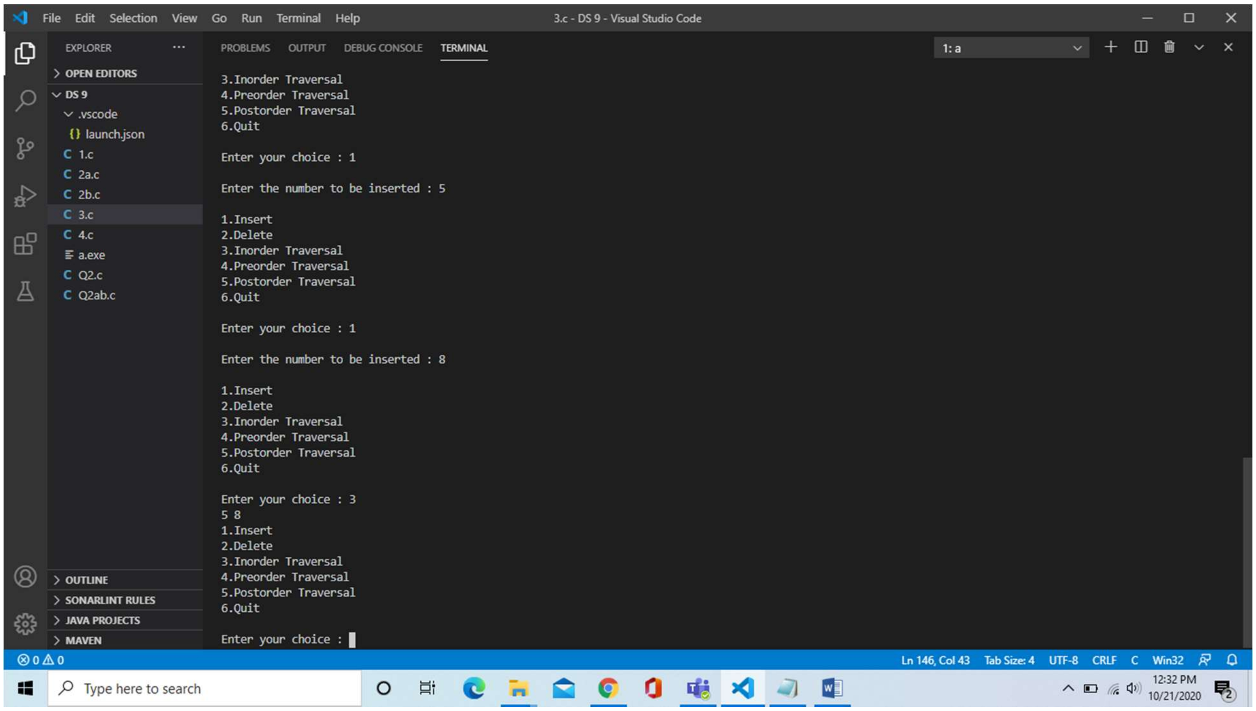
```
                child=ptr->right;


        if(par==NULL )   /*node to be deleted is root node*/
                root=child;
        else if( ptr==par->left) /*node is left child of its parent*/
                par->left=child;
        else                     /*node is right child of its parent*/
                par->right=child;

        s=in_succ(ptr);
        p=in_pred(ptr);

        if(ptr->lthread==false) /*if ptr has left subtree */
                        p->right=s;
        else
        {
                if(ptr->rthread==false) /*if ptr has right subtree*/
                        s->left=p;
        }

        free(ptr);
        return root;
}/*End of case_b( )*/

struct node *case_c(struct node *root, struct node *par,struct node *ptr)
{
        struct node *succ,*parsucc;

        /*Find inorder successor and its parent*/
        parsucc = ptr;
        succ = ptr->right;
        while(succ->left!=NULL)
        {
                parsucc = succ;
                succ = succ->left;
        }

        ptr->info = succ->info;

        if(succ->lthread==true && succ->rthread==true)
                root = case_a(root, parsucc,succ);
        else
                root = case_b(root, parsucc,succ);
        return root;
```



```c
}/*End of case_c( )*/

struct node *in_succ(struct node *ptr)
{
        if(ptr->rthread==true)
                return ptr->right;
        else
        {
                ptr=ptr->right;
                while(ptr->lthread==false)
                        ptr=ptr->left;
                return ptr;
        }
}/*End of in_succ( )*/

struct node *in_pred(struct node *ptr)
{
        if(ptr->lthread==true)
                return ptr->left;
        else
        {
                ptr=ptr->left;
                while(ptr->rthread==false)
                        ptr=ptr->right;
                return ptr;
        }
}/*End of in_pred( )*/

void inorder( struct node *root)
{
        struct node *ptr;
        if(root == NULL )
        {
                printf("Tree is empty");
                return;
        }

        ptr=root;
        /*Find the leftmost node */
        while(ptr->lthread==false)
                ptr=ptr->left;

        while( ptr!=NULL )
        {
                printf("%d ",ptr->info);
```



```
                ptr=in_succ(ptr);
        }
}/*End of inorder( )*/

void preorder(struct node *root )
{
        struct node *ptr;
        if(root==NULL)
        {
                printf("Tree is empty");
                return;
        }
        ptr=root;

        while(ptr!=NULL)
        {
                printf("%d ",ptr->info);
                if(ptr->lthread==false)
                        ptr=ptr->left;
                else if(ptr->rthread==false)
                        ptr=ptr->right;
                else
                {
                        while(ptr!=NULL && ptr->rthread==true)
                                ptr=ptr->right;
                        if(ptr!=NULL)
                                ptr=ptr->right;
                }
        }
}/*End of preorder( )*/
```



```
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Quit

Enter your choice : 1

Enter the number to be inserted : 5

1.Insert
2.Delete
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Quit

Enter your choice : 1

Enter the number to be inserted : 8

1.Insert
2.Delete
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Quit

Enter your choice : 3
5 8
1.Insert
2.Delete
3.Inorder Traversal
4.Preorder Traversal
5.Postorder Traversal
6.Quit

Enter your choice : 
```