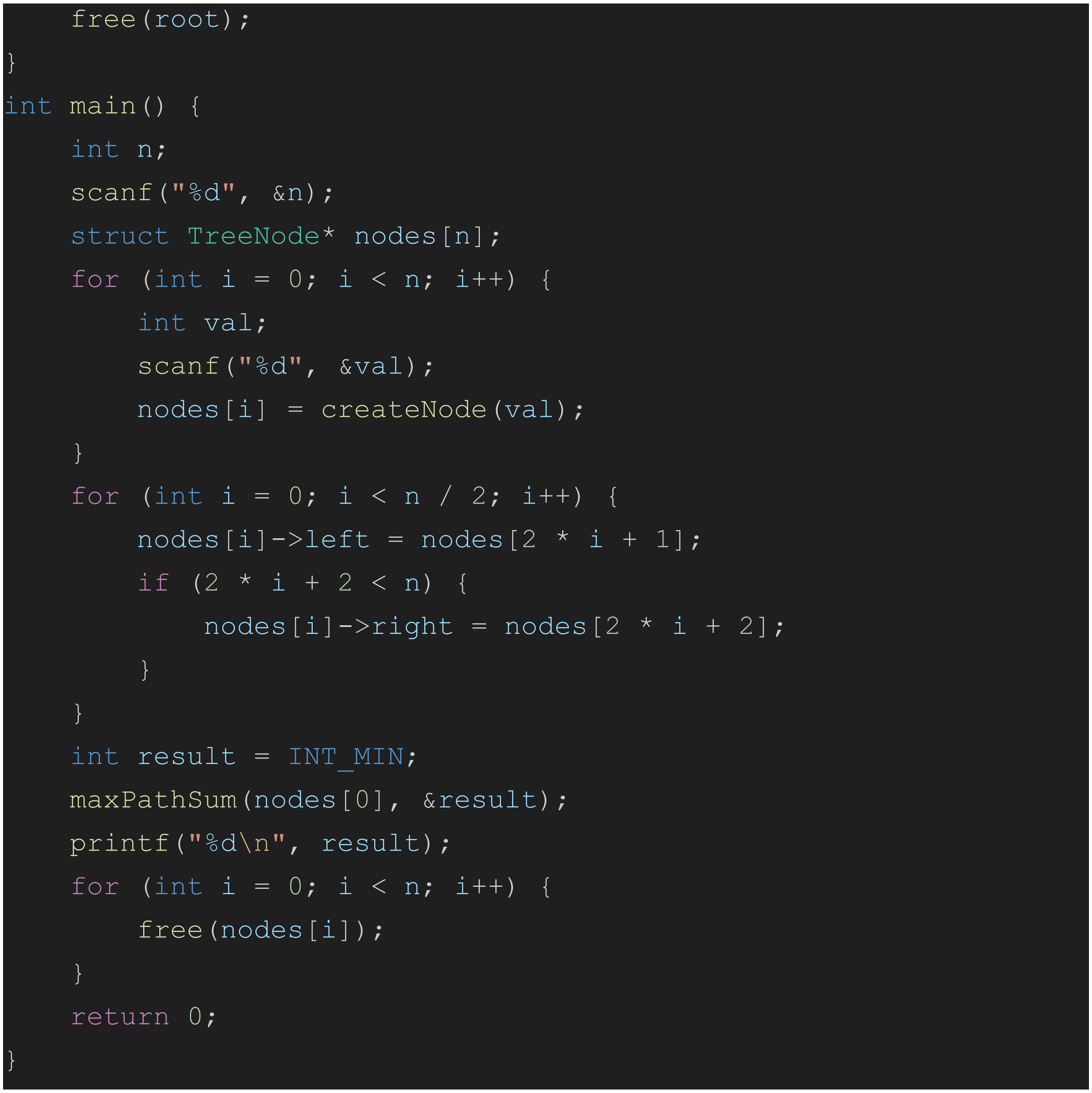
//QA

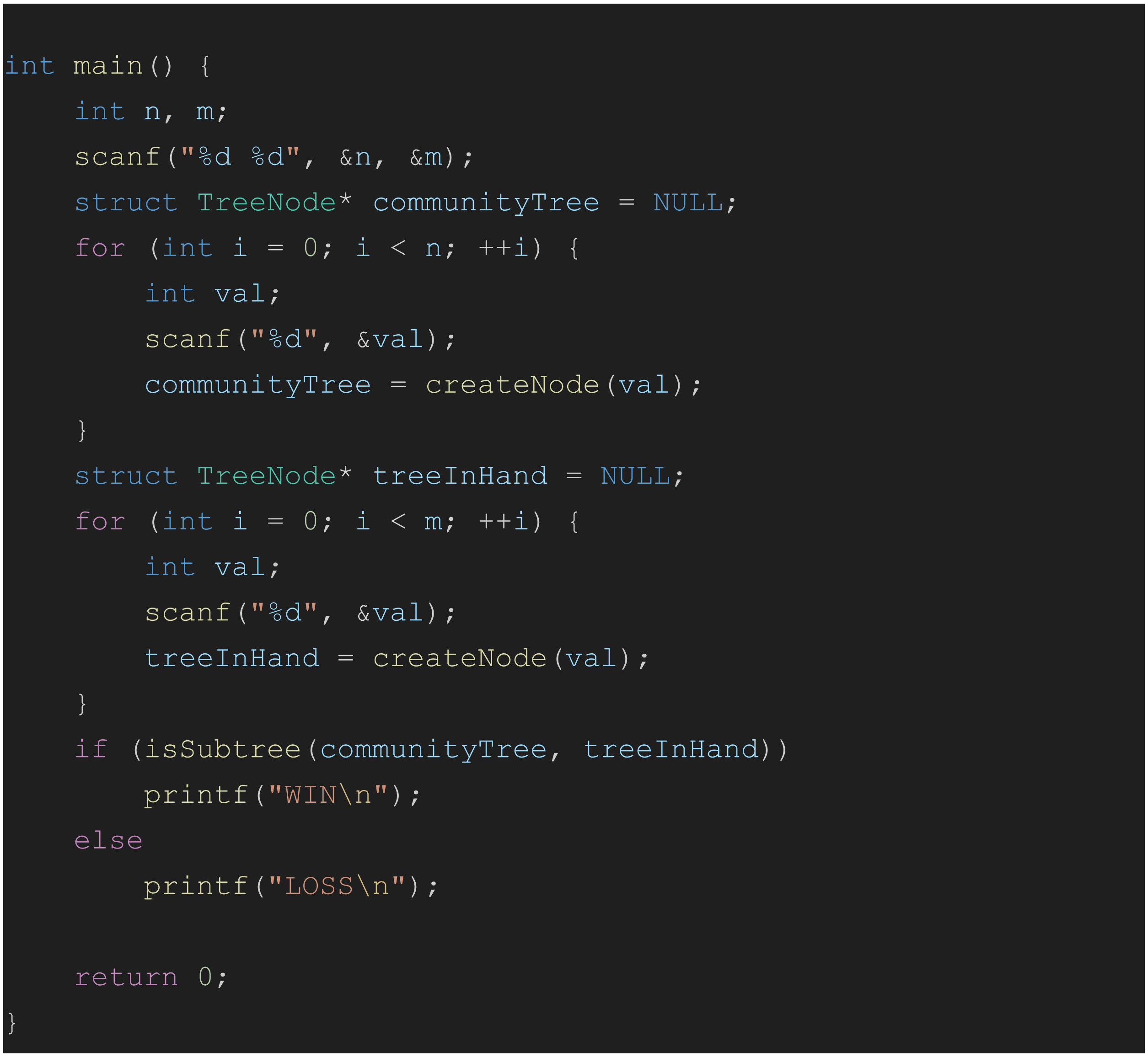
```
#include <stdio.h>
#include <stdlib.h>
#include <limits.h>
#define MAX(x, y) ((x) > (y) ? (x) : (y))
struct TreeNode {
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
};
struct TreeNode* createNode(int val) {
    struct TreeNode* newNode = (struct
TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->val = val;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
int maxPathSum(struct TreeNode* root, int* result) {
    if (root == NULL) return 0;
    int leftSum = maxPathSum(root->left, result);
    int rightSum = maxPathSum(root->right, result);
    int maxEndingHere = MAX(MAX(leftSum, rightSum) + root->val,
root->val);
    int maxThroughHere = MAX(maxEndingHere, leftSum + rightSum +
root->val);
    *result = MAX(*result, maxThroughHere);
    return maxEndingHere;
}
void freeTree(struct TreeNode* root) {
    if (root == NULL) return;
    freeTree(root->left);
    freeTree(root->right);
```



```
    free(root);
}
int main() {
    int n;
    scanf("%d", &n);
    struct TreeNode* nodes[n];
    for (int i = 0; i < n; i++) {
        int val;
        scanf("%d", &val);
        nodes[i] = createNode(val);
    }
    for (int i = 0; i < n / 2; i++) {
        nodes[i]->left = nodes[2 * i + 1];
        if (2 * i + 2 < n) {
            nodes[i]->right = nodes[2 * i + 2];
        }
    }
    int result = INT_MIN;
    maxPathSum(nodes[0], &result);
    printf("%d\n", result);
    for (int i = 0; i < n; i++) {
        free(nodes[i]);
    }
    return 0;
}
```

//QB

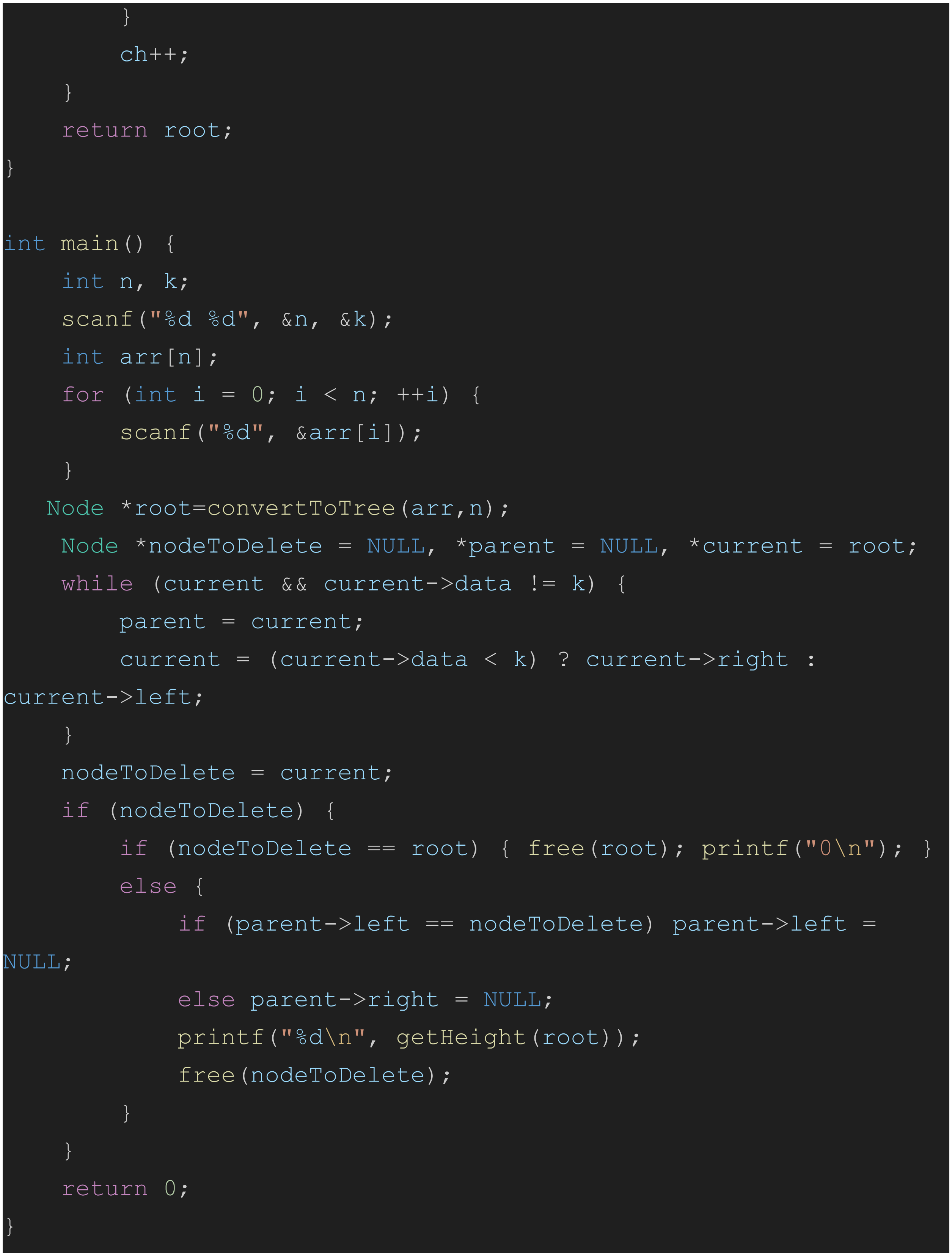
```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* createNode(int val) {
    struct TreeNode* newNode = (struct
TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->val = val;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
int isIdentical(struct TreeNode* tree1, struct TreeNode* tree2)
{
    if (tree1 == NULL && tree2 == NULL)
        return 1;
    if (tree1 == NULL || tree2 == NULL)
        return 0;
    return (tree1->val == tree2->val) &&
           isIdentical(tree1->left, tree2->left) &&
           isIdentical(tree1->right, tree2->right);
}
int isSubtree(struct TreeNode* tree1, struct TreeNode* tree2) {
    if (tree1 == NULL)
        return 0;
    if (isIdentical(tree1, tree2))
        return 1;
    return isSubtree(tree1->left, tree2) ||
isSubtree(tree1->right, tree2);
}
```



```c
int main() {
    int n, m;
    scanf("%d %d", &n, &m);
    struct TreeNode* communityTree = NULL;
    for (int i = 0; i < n; ++i) {
        int val;
        scanf("%d", &val);
        communityTree = createNode(val);
    }
    struct TreeNode* treeInHand = NULL;
    for (int i = 0; i < m; ++i) {
        int val;
        scanf("%d", &val);
        treeInHand = createNode(val);
    }
    if (isSubtree(communityTree, treeInHand))
        printf("WIN\n");
    else
        printf("LOSS\n");

    return 0;
}
```

//QC

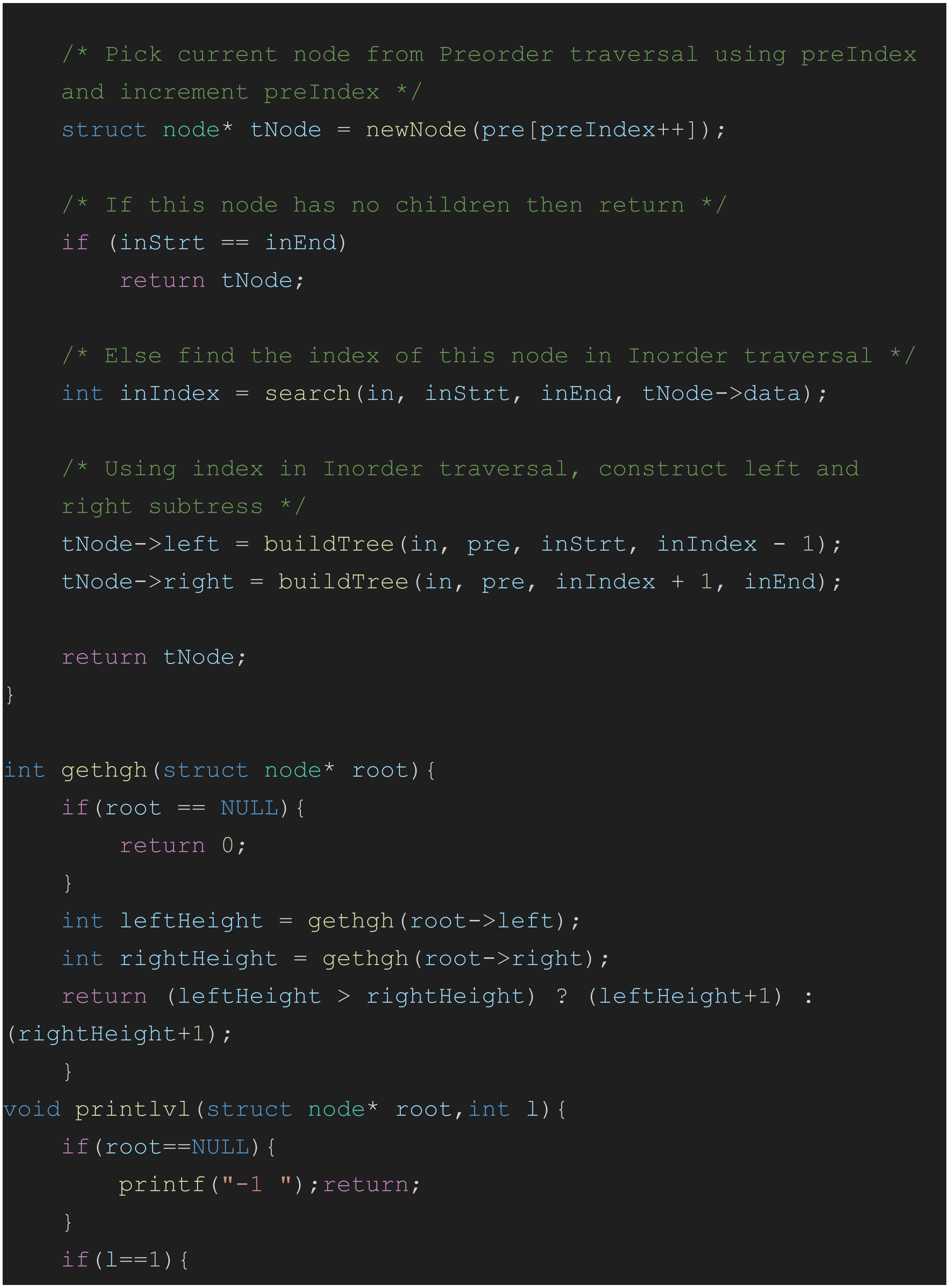
```
#include <stdio.h>
#include <stdlib.h>
typedef struct Node {
    int data;
    struct Node *left, *right;
 } Node;

Node *createNode(int data) {
    Node *newNode = malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
int getHeight(Node *root) {
    return root ? (getHeight(root->left) >
getHeight(root->right) ? getHeight(root->left) :
getHeight(root->right)) + 1 : 0;
}
Node* convertToTree(int arr[], int n) {
     Node* root = createNode(arr[0]);
     Node* queue[n];
    int i = 0, ch = 1;
    queue[i++] = root;
    while (ch < n) {
         Node* node = queue[ch / 2];
        if (arr[ch] != -1) {
            node->left = createNode(arr[ch]);
            queue[i++] = node->left;
        }
        ch++;
        if (arr[ch] != -1 && ch < n) {
            node->right = createNode(arr[ch]);
            queue[i++] = node->right;
```



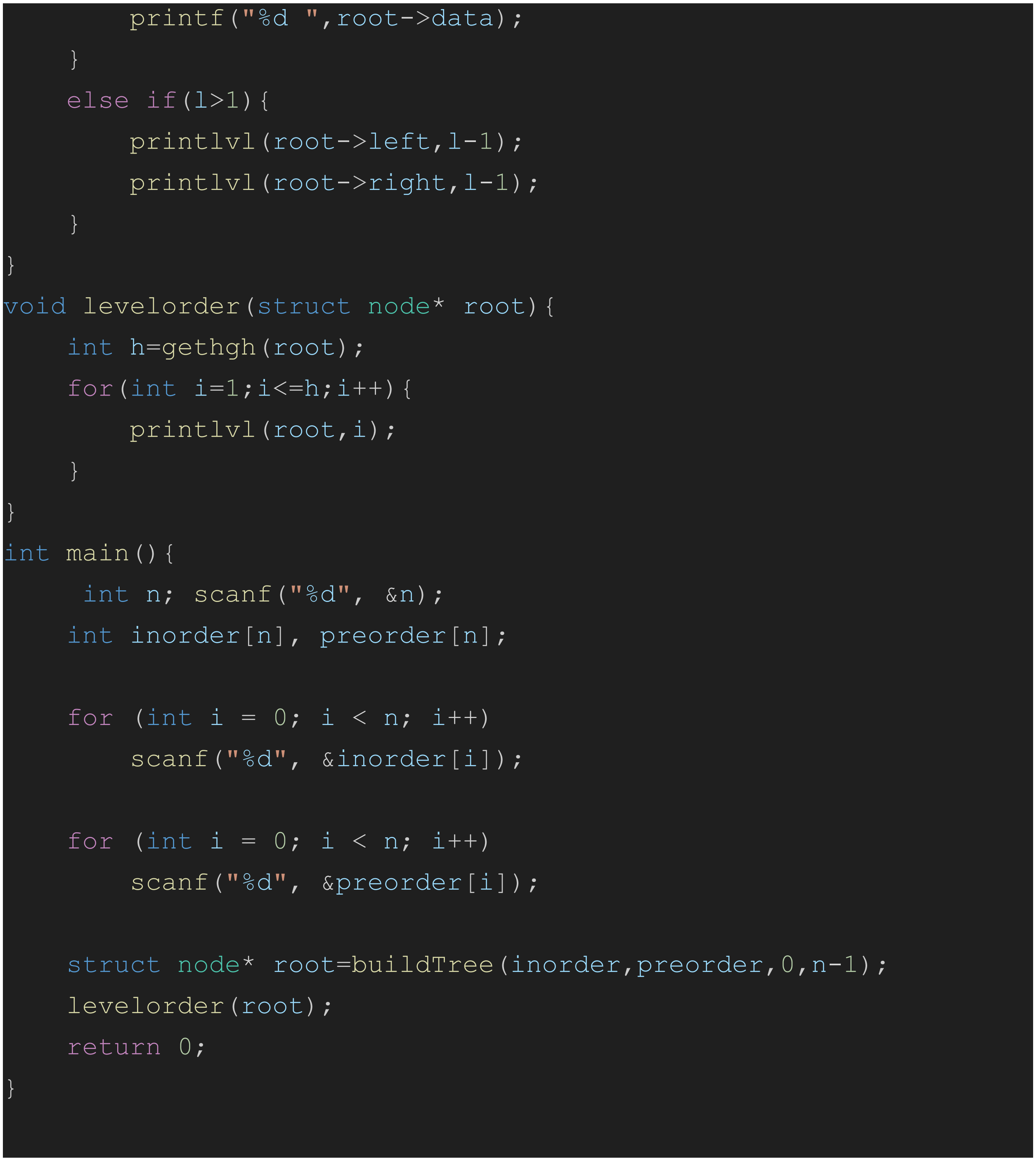
```
        }
        ch++;
    }
    return root;
}

int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    int arr[n];
    for (int i = 0; i < n; ++i) {
        scanf("%d", &arr[i]);
    }
   Node *root=convertToTree(arr,n);
    Node *nodeToDelete = NULL, *parent = NULL, *current = root;
    while (current && current->data != k) {
        parent = current;
        current = (current->data < k) ? current->right :
current->left;
    }
    nodeToDelete = current;
    if (nodeToDelete) {
        if (nodeToDelete == root) { free(root); printf("0\n"); }
        else {
            if (parent->left == nodeToDelete) parent->left =
NULL;
            else parent->right = NULL;
            printf("%d\n", getHeight(root));
            free(nodeToDelete);
        }
    }
    return 0;
}
```

//QD

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

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

struct node* newNode(int data){
    struct node* node = (struct node*)malloc(sizeof(struct
node));
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    return (node);
}

int search(int arr[], int strt, int end, char value){
    int i;
    for (i = strt; i <= end; i++) {
        if (arr[i] == value)
            return i;
    }
}

struct node* buildTree(int in[], int pre[], int inStrt, int
inEnd){
    static int preIndex = 0;

    if (inStrt > inEnd)
        return NULL;
```



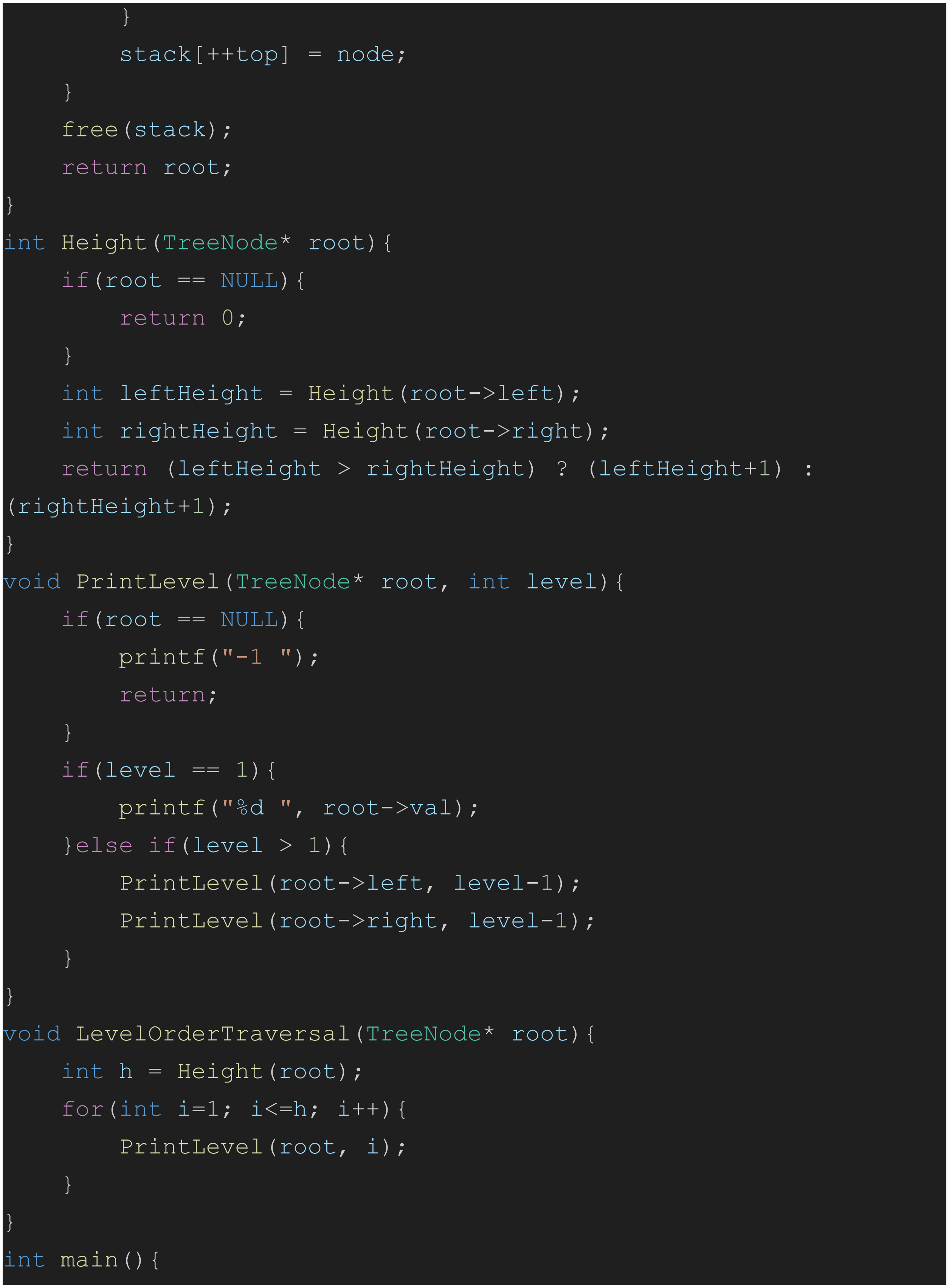
```
    /* Pick current node from Preorder traversal using preIndex
    and increment preIndex */
    struct node* tNode = newNode(pre[preIndex++]);

    /* If this node has no children then return */
    if (inStrt == inEnd)
        return tNode;

    /* Else find the index of this node in Inorder traversal */
    int inIndex = search(in, inStrt, inEnd, tNode->data);

    /* Using index in Inorder traversal, construct left and
    right subtress */
    tNode->left = buildTree(in, pre, inStrt, inIndex - 1);
    tNode->right = buildTree(in, pre, inIndex + 1, inEnd);

    return tNode;
}

int gethgh(struct node* root){
    if(root == NULL){
        return 0;
    }
    int leftHeight = gethgh(root->left);
    int rightHeight = gethgh(root->right);
    return (leftHeight > rightHeight) ? (leftHeight+1) :
(rightHeight+1);
    }
void printlvl(struct node* root,int l){
    if(root==NULL){
        printf("-1 ");return;
    }
    if(l==1){
```



```
        printf("%d ",root->data);
    }
    else if(l>1){
        printlvl(root->left,l-1);
        printlvl(root->right,l-1);
    }
}
void levelorder(struct node* root){
    int h=gethgh(root);
    for(int i=1;i<=h;i++){
        printlvl(root,i);
    }
}
int main(){
     int n; scanf("%d", &n);
    int inorder[n], preorder[n];

    for (int i = 0; i < n; i++)
        scanf("%d", &inorder[i]);

    for (int i = 0; i < n; i++)
        scanf("%d", &preorder[i]);

    struct node* root=buildTree(inorder,preorder,0,n-1);
    levelorder(root);
    return 0;
}
```

//QE

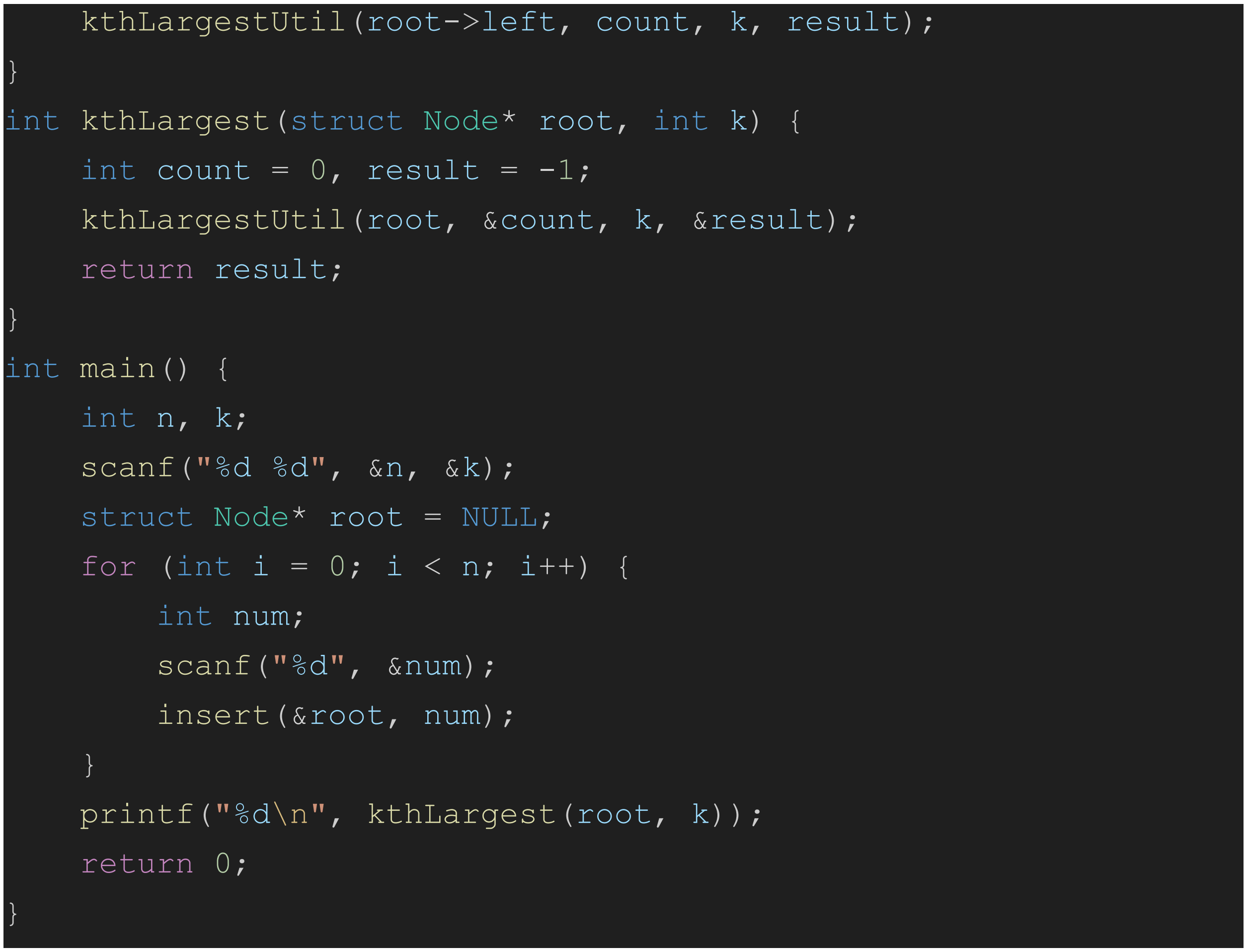
```
#include<stdio.h>
#include<stdlib.h>
typedef struct TreeNode{
    int val;
    struct TreeNode *left;
    struct TreeNode *right;
} TreeNode;
TreeNode* CreateNode(int value){
    struct TreeNode* node = (struct
TreeNode*)malloc(sizeof(struct TreeNode));
    node->val = value;
    node->left = NULL;
    node->right = NULL;
    return node;
}
TreeNode* BuildBST(int n, int postorder[]){
    if(n<=0){
        return NULL;
    }
    TreeNode** stack = (TreeNode**)malloc(n * sizeof(TreeNode));
    int top = -1;
    TreeNode* root = CreateNode(postorder[n - 1]);
    stack[++top] = root;
    for(int i = n-2 ; i>=0; i--){
        TreeNode* node = CreateNode(postorder[i]);
        if(postorder[i] > stack[top]->val){
            stack[top]->right = node;
        }else{
            TreeNode* parent = NULL;
            while(top >= 0 && postorder[i] < stack[top]->val){
                parent = stack[top--];
            }
            parent->left = node;
```



```
        }
        stack[++top] = node;
    }
    free(stack);
    return root;
}
int Height(TreeNode* root){
    if(root == NULL){
        return 0;
    }
    int leftHeight = Height(root->left);
    int rightHeight = Height(root->right);
    return (leftHeight > rightHeight) ? (leftHeight+1) :
(rightHeight+1);
}
void PrintLevel(TreeNode* root, int level){
    if(root == NULL){
        printf("-1 ");
        return;
    }
    if(level == 1){
        printf("%d ", root->val);
    }else if(level > 1){
        PrintLevel(root->left, level-1);
        PrintLevel(root->right, level-1);
    }
}
void LevelOrderTraversal(TreeNode* root){
    int h = Height(root);
    for(int i=1; i<=h; i++){
        PrintLevel(root, i);
    }
}
int main(){
    int n;
    scanf("%d",&n);
    int postorder[n];
    for(int i=0; i<n; i++){
        scanf("%d",&postorder[i]);
    }
    TreeNode* root = BuildBST(n, postorder);
    LevelOrderTraversal(root);
    return 0;
}
```

//QF

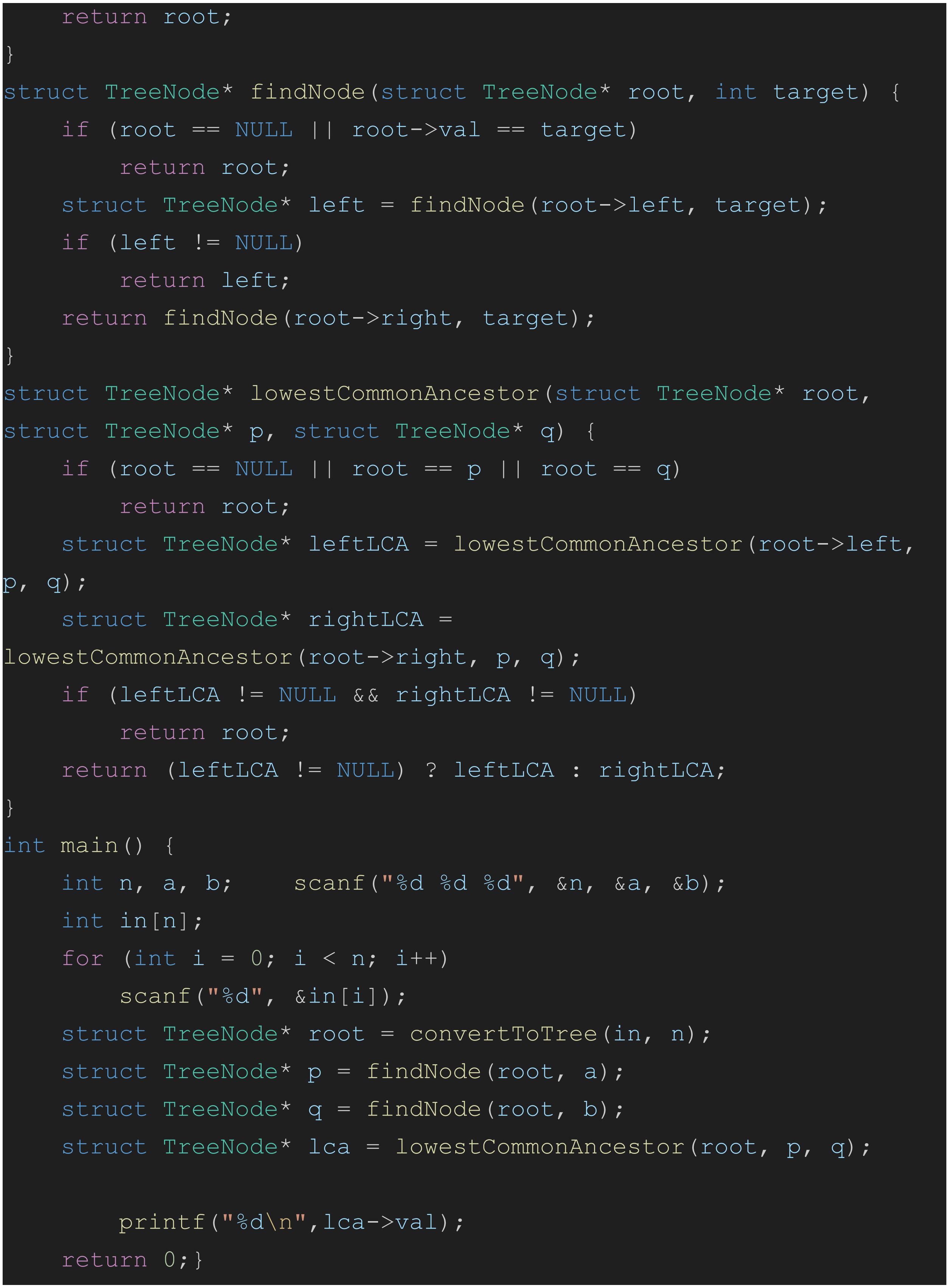
```
 #include <stdio.h>
#include <stdlib.h>
struct Node {
    int data;
    struct Node *left, *right;
};
struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
void insert(struct Node** root, int data) {
    if (*root == NULL) {
        *root = createNode(data);
        return;
    }
    if (data < (*root)->data)
        insert(&((*root)->left), data);
    else
        insert(&((*root)->right), data);
}
void kthLargestUtil(struct Node* root, int* count, int k, int* result) {
    if (root == NULL || *count >= k)
        return;
    kthLargestUtil(root->right, count, k, result);
    (*count)++;
    if (*count == k) {
        *result = root->data;
        return;
    }
```



```c
    kthLargestUtil(root->left, count, k, result);
}
int kthLargest(struct Node* root, int k) {
    int count = 0, result = -1;
    kthLargestUtil(root, &count, k, &result);
    return result;
}
int main() {
    int n, k;
    scanf("%d %d", &n, &k);
    struct Node* root = NULL;
    for (int i = 0; i < n; i++) {
        int num;
        scanf("%d", &num);
        insert(&root, num);
    }
    printf("%d\n", kthLargest(root, k));
    return 0;
}
```

//QG

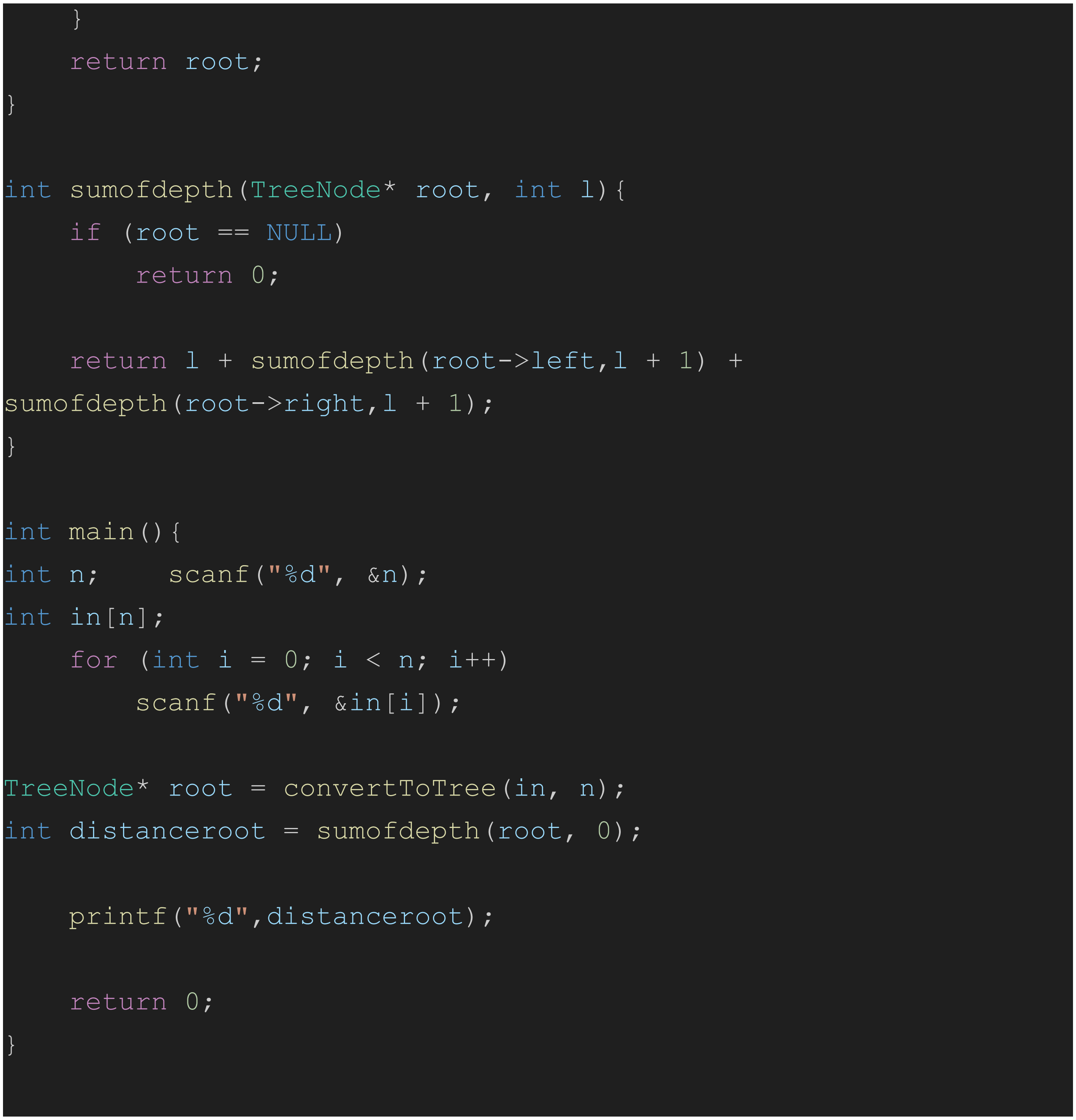
```
#include <stdio.h>
#include <stdlib.h>
struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
};
struct TreeNode* createNode(int val) {
    struct TreeNode* newNode = (struct
TreeNode*)malloc(sizeof(struct TreeNode));
    newNode->val = val;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
struct TreeNode* convertToTree(int arr[], int n) {
    struct TreeNode* root = createNode(arr[0]);
    struct TreeNode* queue[n];
    int i = 0, ch = 1;
    queue[i++] = root;
    while (ch < n) {
        struct TreeNode* node = queue[ch / 2];
        if (arr[ch] != -1) {
            node->left = createNode(arr[ch]);
            queue[i++] = node->left;
        }
        ch++;
        if (arr[ch] != -1 && ch < n) {
            node->right = createNode(arr[ch]);
            queue[i++] = node->right;
        }
        ch++;
    }
```



```
    return root;
}
struct TreeNode* findNode(struct TreeNode* root, int target) {
    if (root == NULL || root->val == target)
        return root;
    struct TreeNode* left = findNode(root->left, target);
    if (left != NULL)
        return left;
    return findNode(root->right, target);
}
struct TreeNode* lowestCommonAncestor(struct TreeNode* root,
struct TreeNode* p, struct TreeNode* q) {
    if (root == NULL || root == p || root == q)
        return root;
    struct TreeNode* leftLCA = lowestCommonAncestor(root->left,
p, q);
    struct TreeNode* rightLCA =
lowestCommonAncestor(root->right, p, q);
    if (leftLCA != NULL && rightLCA != NULL)
        return root;
    return (leftLCA != NULL) ? leftLCA : rightLCA;
}
int main() {
    int n, a, b;    scanf("%d %d %d", &n, &a, &b);
    int in[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &in[i]);
    struct TreeNode* root = convertToTree(in, n);
    struct TreeNode* p = findNode(root, a);
    struct TreeNode* q = findNode(root, b);
    struct TreeNode* lca = lowestCommonAncestor(root, p, q);

        printf("%d\n",lca->val);
    return 0;}
```

//QH

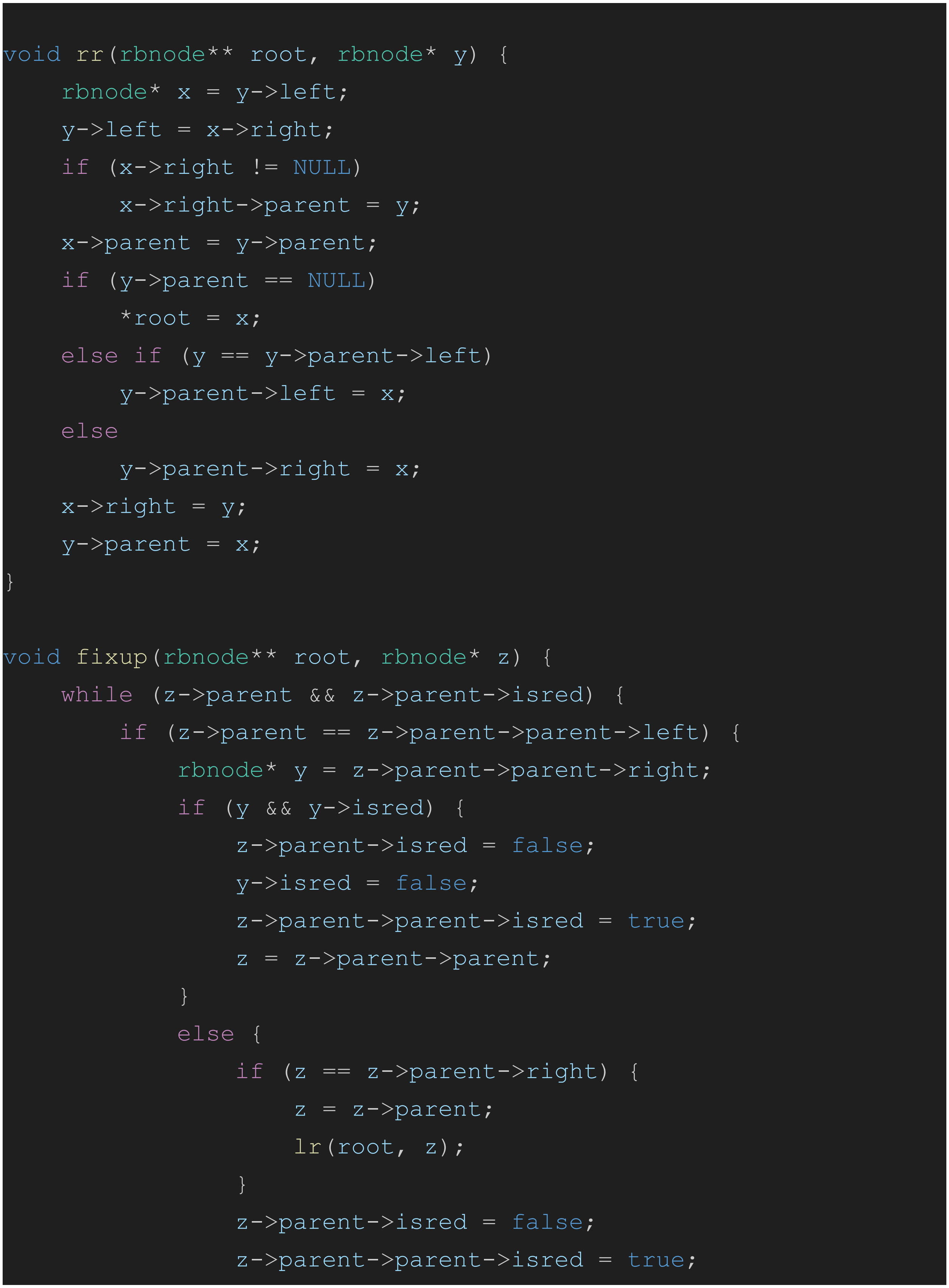
```
#include <stdio.h>
#include <stdlib.h>

typedef struct TreeNode {
    int val;
    struct TreeNode* left;
    struct TreeNode* right;
}TreeNode;

TreeNode* createNode(int val) {
     TreeNode* newNode = ( TreeNode*)malloc(sizeof( TreeNode));
    newNode->val = val;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}
TreeNode* convertToTree(int arr[], int n) {
     TreeNode* root = createNode(arr[0]);
     TreeNode* queue[n];
    int i = 0, ch = 1;
    queue[i++] = root;
    while (ch < n) {
         TreeNode* node = queue[ch / 2];
        if (arr[ch] != -1) {
            node->left = createNode(arr[ch]);
            queue[i++] = node->left;
        }
        ch++;
        if (arr[ch] != -1 && ch < n) {
            node->right = createNode(arr[ch]);
            queue[i++] = node->right;
        }
        ch++;
```



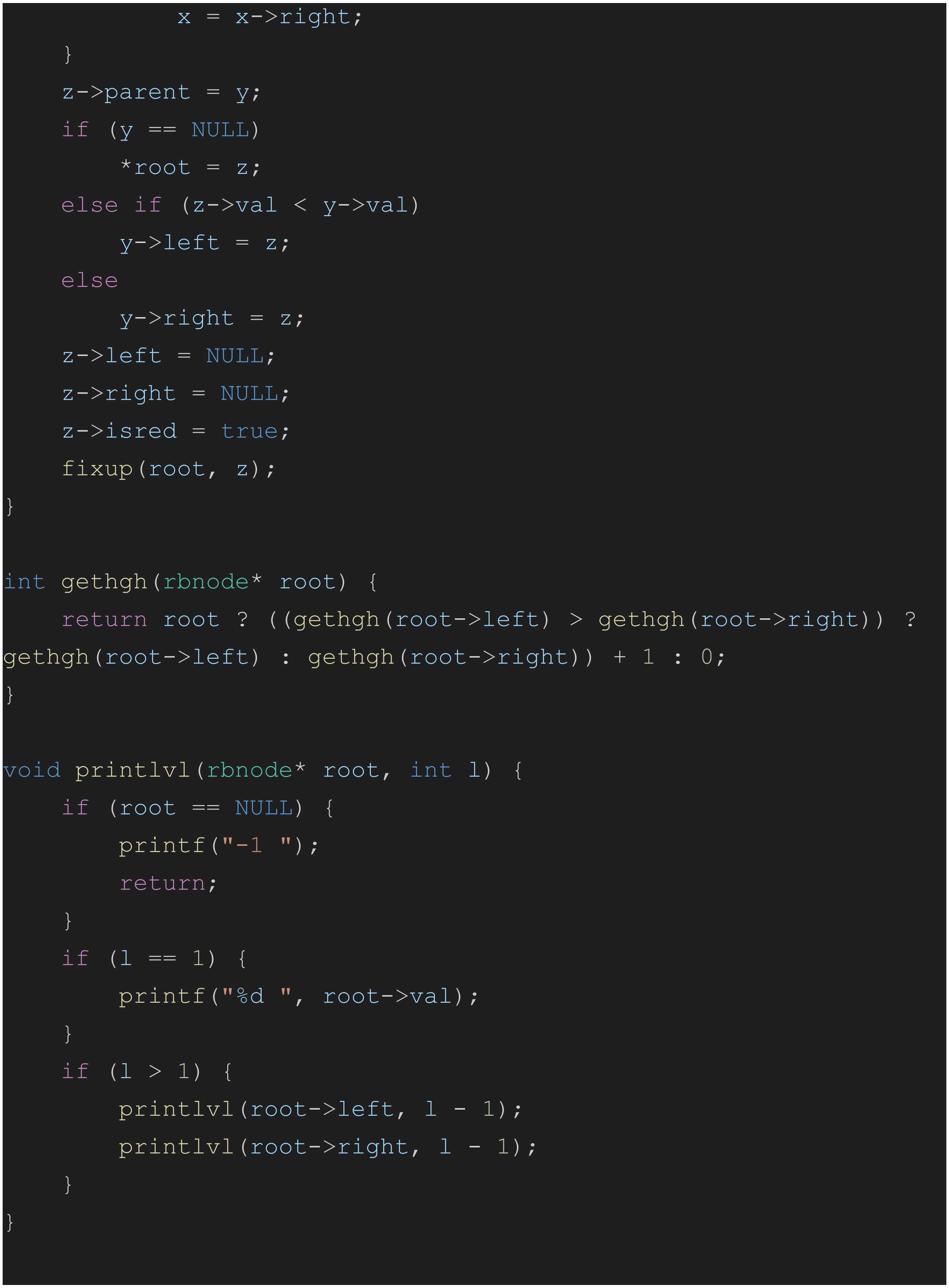
```
    }
    return root;
}

int sumofdepth(TreeNode* root, int l){
    if (root == NULL)
        return 0;

    return l + sumofdepth(root->left,l + 1) +
sumofdepth(root->right,l + 1);
}

int main(){
int n;    scanf("%d", &n);
int in[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &in[i]);

TreeNode* root = convertToTree(in, n);
int distanceroot = sumofdepth(root, 0);

    printf("%d",distanceroot);

    return 0;
}
```

//QI

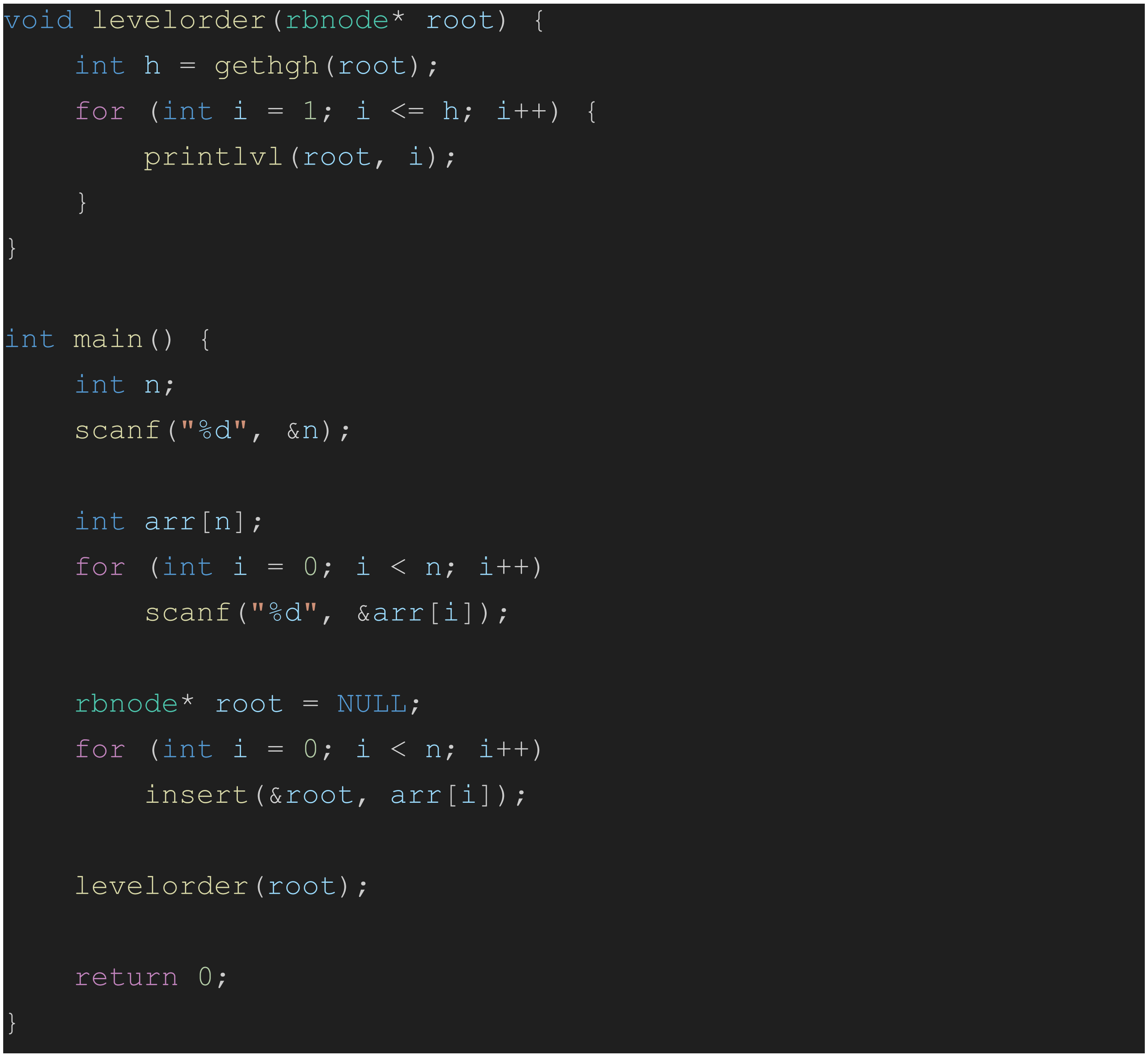
```c
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

typedef struct rbnode {
    int val;
    struct rbnode* left, *right, *parent;
    bool isred;
} rbnode;

rbnode* create(int x) {
    rbnode* temp = (rbnode*)malloc(sizeof(rbnode));
    temp->isred = true;
    temp->val = x;
    temp->left = temp->right = temp->parent = NULL;
    return temp;
}

void lr(rbnode** root, rbnode* x) {
    rbnode* y = x->right;
    x->right = y->left;
    if (y->left != NULL)
        y->left->parent = x;
    y->parent = x->parent;
    if (x->parent == NULL)
        *root = y;
    else if (x == x->parent->left)
        x->parent->left = y;
    else
        x->parent->right = y;
    y->left = x;
    x->parent = y;
}
```



```
void rr(rbnode** root, rbnode* y) {
    rbnode* x = y->left;
    y->left = x->right;
    if (x->right != NULL)
        x->right->parent = y;
    x->parent = y->parent;
    if (y->parent == NULL)
        *root = x;
    else if (y == y->parent->left)
        y->parent->left = x;
    else
        y->parent->right = x;
    x->right = y;
    y->parent = x;
}

void fixup(rbnode** root, rbnode* z) {
    while (z->parent && z->parent->isred) {
        if (z->parent == z->parent->parent->left) {
            rbnode* y = z->parent->parent->right;
            if (y && y->isred) {
                z->parent->isred = false;
                y->isred = false;
                z->parent->parent->isred = true;
                z = z->parent->parent;
            }
            else {
                if (z == z->parent->right) {
                    z = z->parent;
                    lr(root, z);
                }
                z->parent->isred = false;
                z->parent->parent->isred = true;
                rr(root, z->parent->parent);
            }
        }
        else {
            rbnode* y = z->parent->parent->left;
            if (y && y->isred) {
                z->parent->isred = false;
                y->isred = false;
                z->parent->parent->isred = true;
                z = z->parent->parent;
            }
            else {
                if (z == z->parent->left) {
                    z = z->parent;
                    rr(root, z);
                }
                z->parent->isred = false;
                z->parent->parent->isred = true;
                lr(root, z->parent->parent);
            }
        }
    }
    (*root)->isred = false;
}

void insert(rbnode** root, int data) {
    rbnode* z = create(data);
    rbnode* y = NULL;
    rbnode* x = *root;
    while (x != NULL) {
        y = x;
        if (z->val < x->val)
            x = x->left;
        else
```



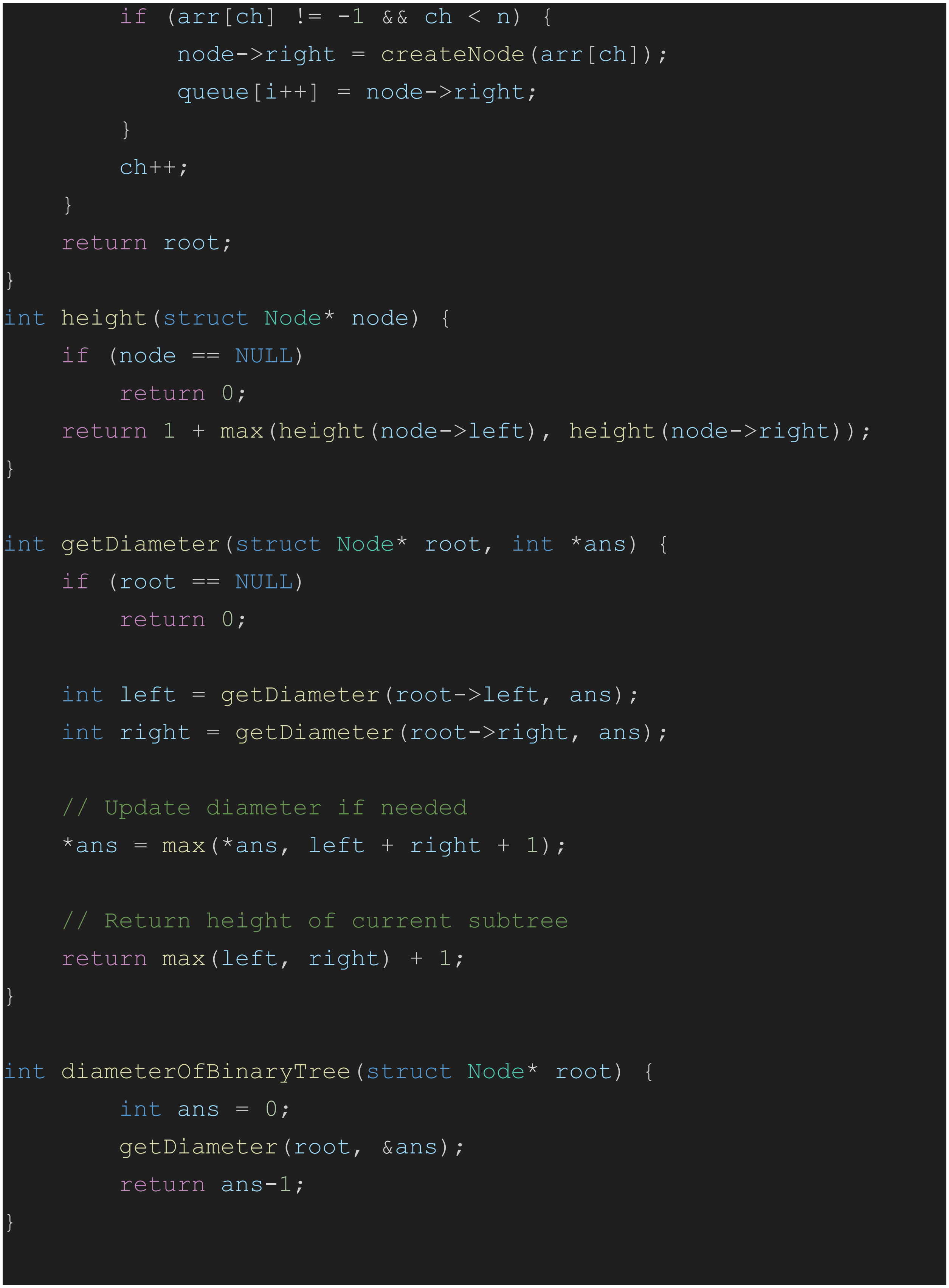
```
            x = x->right;
    }
    z->parent = y;
    if (y == NULL)
        *root = z;
    else if (z->val < y->val)
        y->left = z;
    else
        y->right = z;
    z->left = NULL;
    z->right = NULL;
    z->isred = true;
    fixup(root, z);
}

int gethgh(rbnode* root) {
    return root ? ((gethgh(root->left) > gethgh(root->right)) ?
gethgh(root->left) : gethgh(root->right)) + 1 : 0;
}

void printlvl(rbnode* root, int l) {
    if (root == NULL) {
        printf("-1 ");
        return;
    }
    if (l == 1) {
        printf("%d ", root->val);
    }
    if (l > 1) {
        printlvl(root->left, l - 1);
        printlvl(root->right, l - 1);
    }
}
```



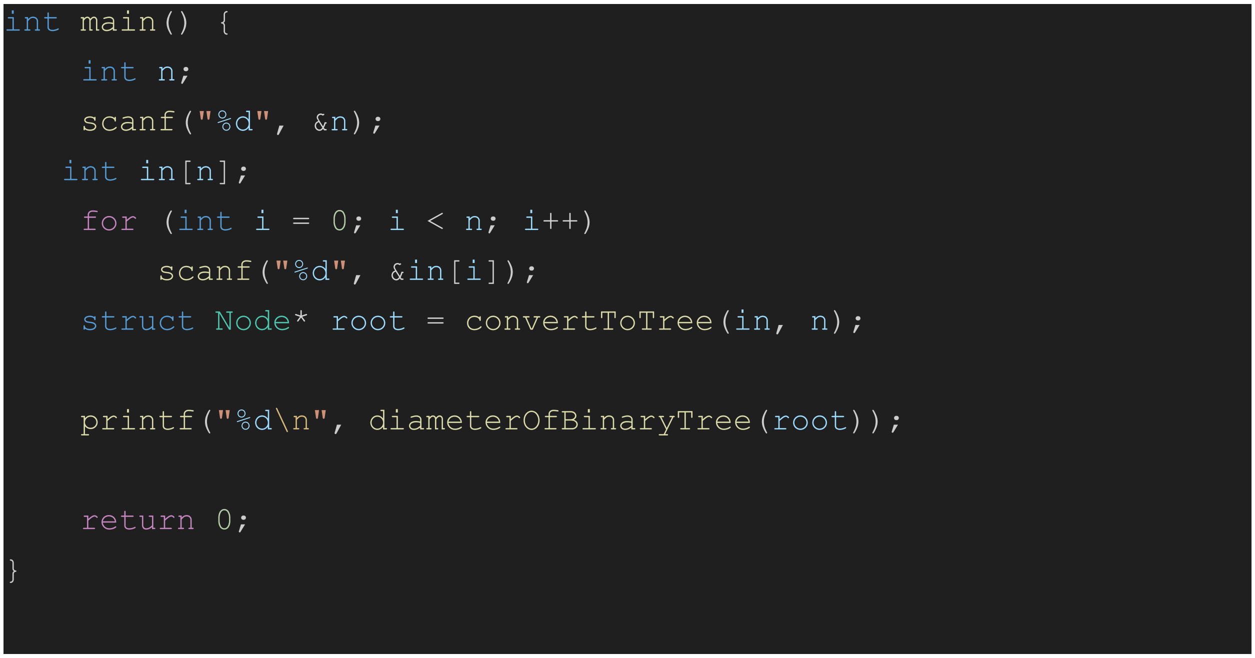
```c
void levelorder(rbnode* root) {
    int h = gethgh(root);
    for (int i = 1; i <= h; i++) {
        printlvl(root, i);
    }
}

int main() {
    int n;
    scanf("%d", &n);

    int arr[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &arr[i]);

    rbnode* root = NULL;
    for (int i = 0; i < n; i++)
        insert(&root, arr[i]);

    levelorder(root);

    return 0;
}
```

//QJ

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

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

struct Node* createNode(int data) {
    struct Node* newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;
    return newNode;
}

int max(int a, int b) {
    return (a > b) ? a : b;
}
struct Node* convertToTree(int arr[], int n) {
    struct Node* root = createNode(arr[0]);
    struct Node* queue[n];
    int i = 0, ch = 1;
    queue[i++] = root;
    while (ch < n) {
        struct Node* node = queue[ch / 2];
        if (arr[ch] != -1) {
            node->left = createNode(arr[ch]);
            queue[i++] = node->left;
        }
        ch++;
```



```
        if (arr[ch] != -1 && ch < n) {
            node->right = createNode(arr[ch]);
            queue[i++] = node->right;
        }
        ch++;
    }
    return root;
}
int height(struct Node* node) {
    if (node == NULL)
        return 0;
    return 1 + max(height(node->left), height(node->right));
}

int getDiameter(struct Node* root, int *ans) {
    if (root == NULL)
        return 0;

    int left = getDiameter(root->left, ans);
    int right = getDiameter(root->right, ans);

    // Update diameter if needed
    *ans = max(*ans, left + right + 1);

    // Return height of current subtree
    return max(left, right) + 1;
}

int diameterOfBinaryTree(struct Node* root) {
        int ans = 0;
        getDiameter(root, &ans);
        return ans-1;
}
```



```c
int main() {
    int n;
    scanf("%d", &n);
   int in[n];
    for (int i = 0; i < n; i++)
        scanf("%d", &in[i]);
    struct Node* root = convertToTree(in, n);

    printf("%d\n", diameterOfBinaryTree(root));

    return 0;
}
```