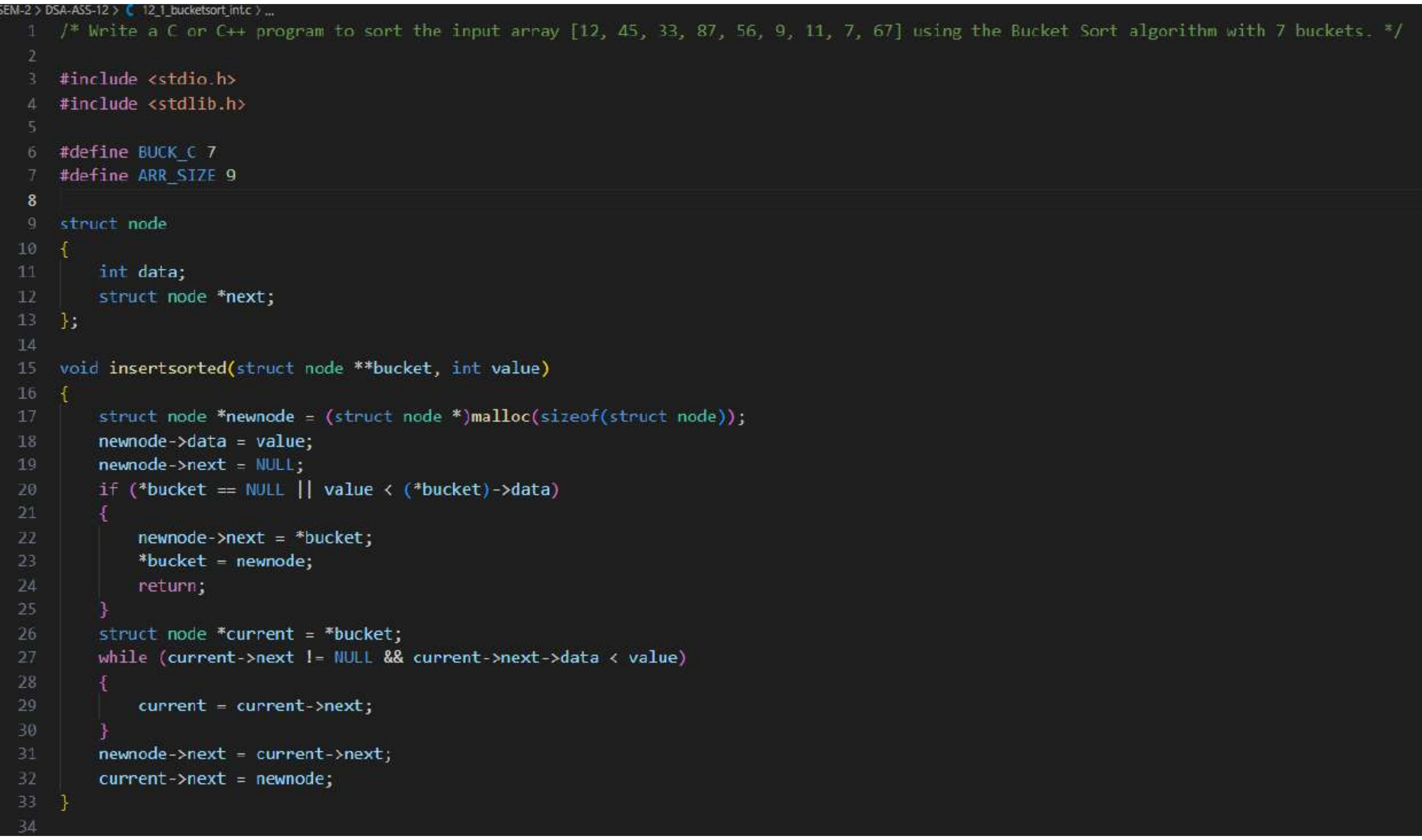
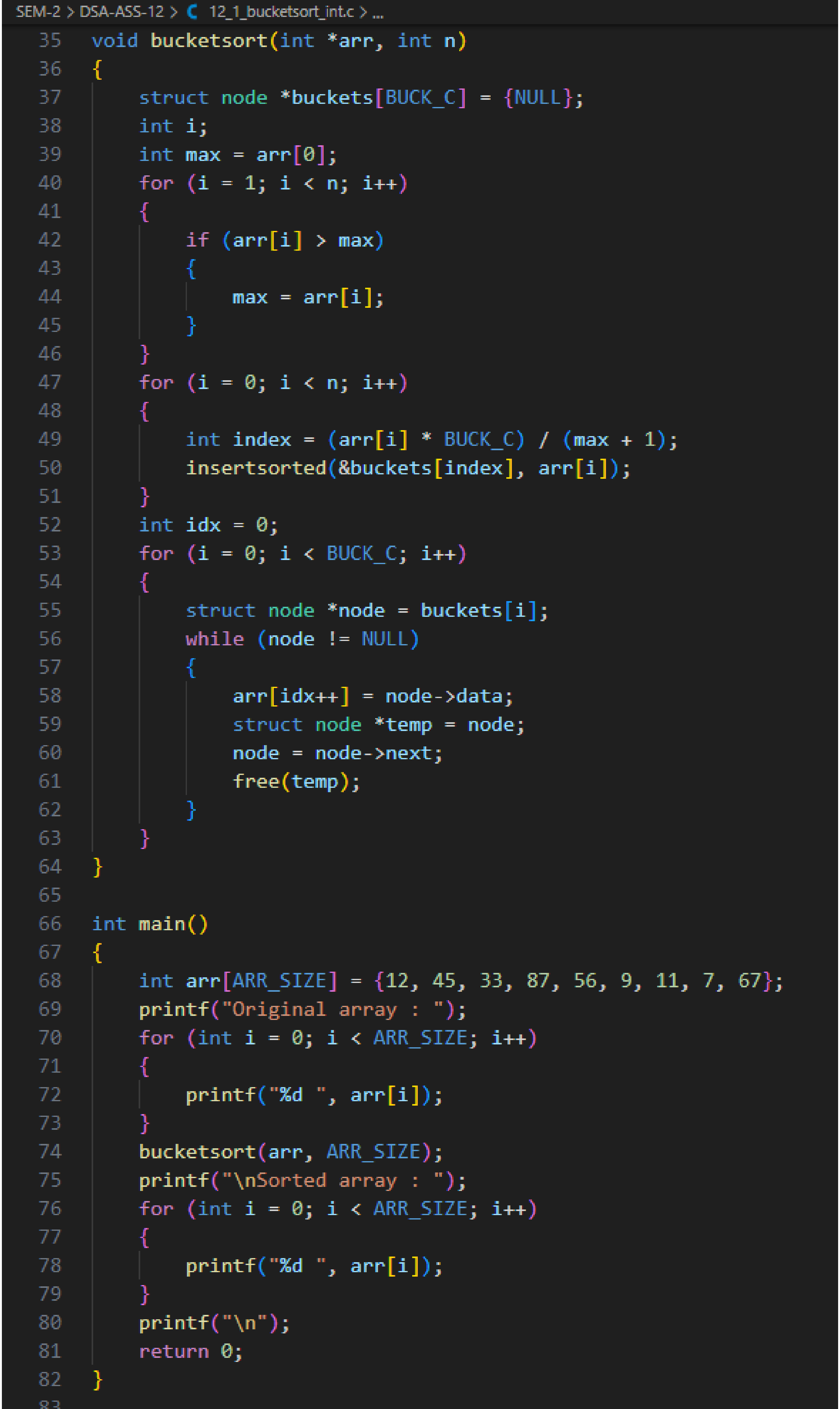
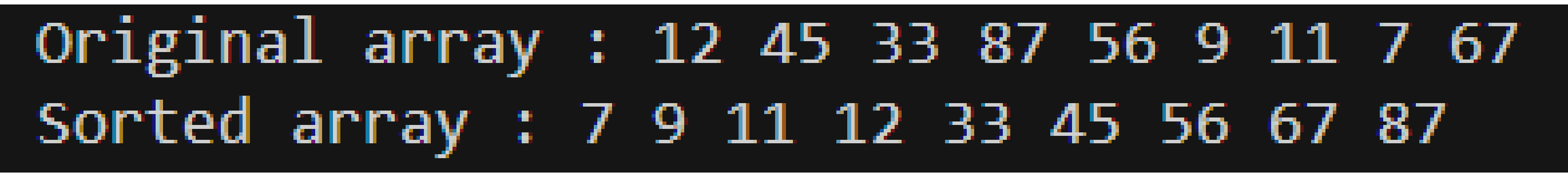


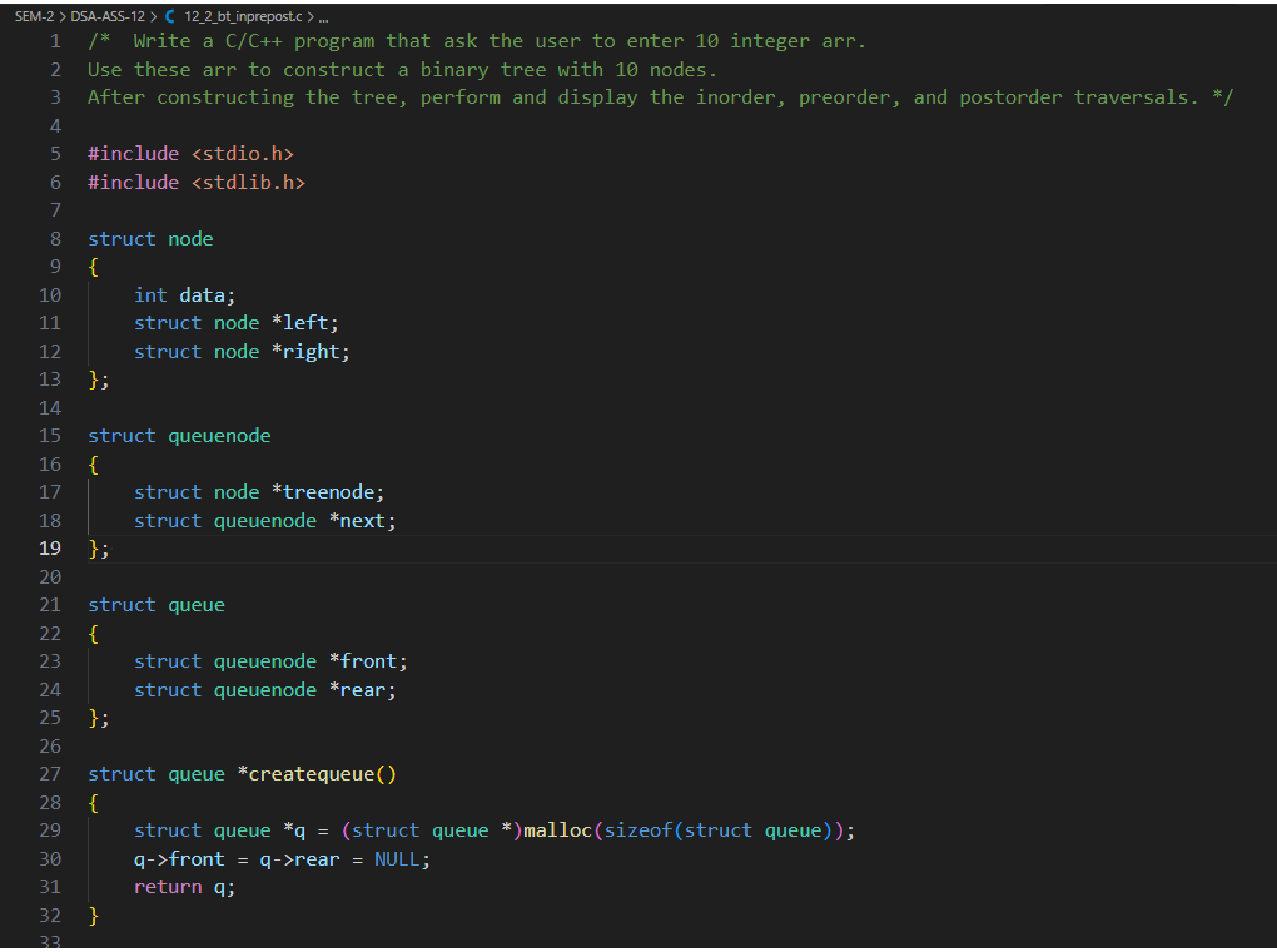
```c
/* Write a C or C++ program to sort the input array [12, 45, 33, 87, 56, 9, 11, 7, 67] using the Bucket Sort algorithm with 7 buckets. */

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

#define BUCK_C 7
#define ARR_SIZE 9

struct node
{
    int data;
    struct node *next;
};

void insertsorted(struct node **bucket, int value)
{
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = value;
    newnode->next = NULL;
    if (*bucket == NULL || value < (*bucket)->data)
    {
        newnode->next = *bucket;
        *bucket = newnode;
        return;
    }
    struct node *current = *bucket;
    while (current->next != NULL && current->next->data < value)
    {
        current = current->next;
    }
    newnode->next = current->next;
    current->next = newnode;
}

```



```c
void bucketsort(int *arr, int n)
{
    struct node *buckets[BUCK_C] = {NULL};
    int i;
    int max = arr[0];
    for (i = 1; i < n; i++)
    {
        if (arr[i] > max)
        {
            max = arr[i];
        }
    }
    for (i = 0; i < n; i++)
    {
        int index = (arr[i] * BUCK_C) / (max + 1);
        insertsorted(&buckets[index], arr[i]);
    }
    int idx = 0;
    for (i = 0; i < BUCK_C; i++)
    {
        struct node *node = buckets[i];
        while (node != NULL)
        {
            arr[idx++] = node->data;
            struct node *temp = node;
            node = node->next;
            free(temp);
        }
    }
}

int main()
{
    int arr[ARR_SIZE] = {12, 45, 33, 87, 56, 9, 11, 7, 67};
    printf("Original array : ");
    for (int i = 0; i < ARR_SIZE; i++)
    {
        printf("%d ", arr[i]);
    }
    bucketsort(arr, ARR_SIZE);
    printf("\nSorted array : ");
    for (int i = 0; i < ARR_SIZE; i++)
    {
        printf("%d ", arr[i]);
    }
    printf("\n");
    return 0;
}
```

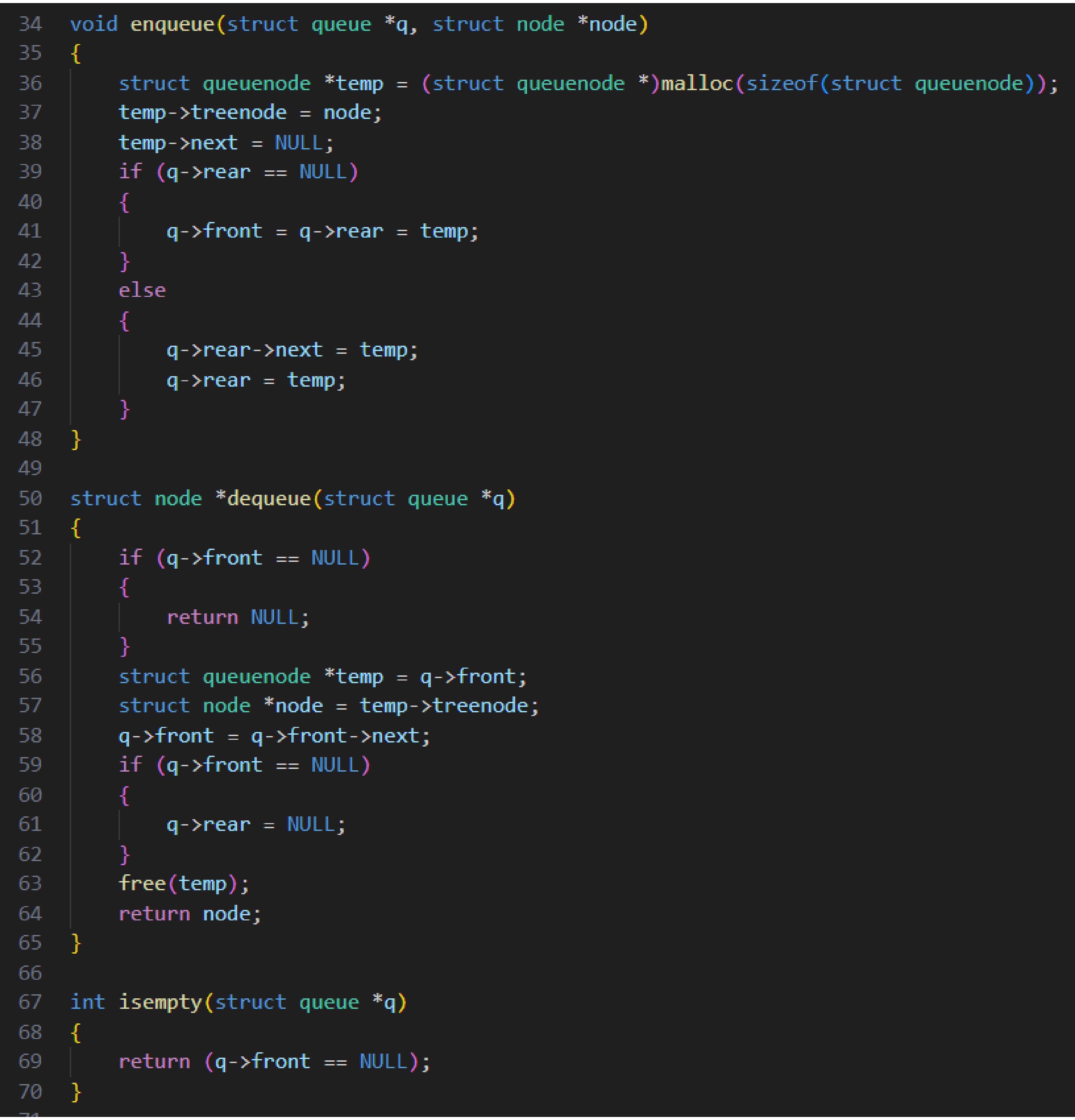


```
Original array : 12 45 33 87 56 9 11 7 67
Sorted array : 7 9 11 12 33 45 56 67 87
```

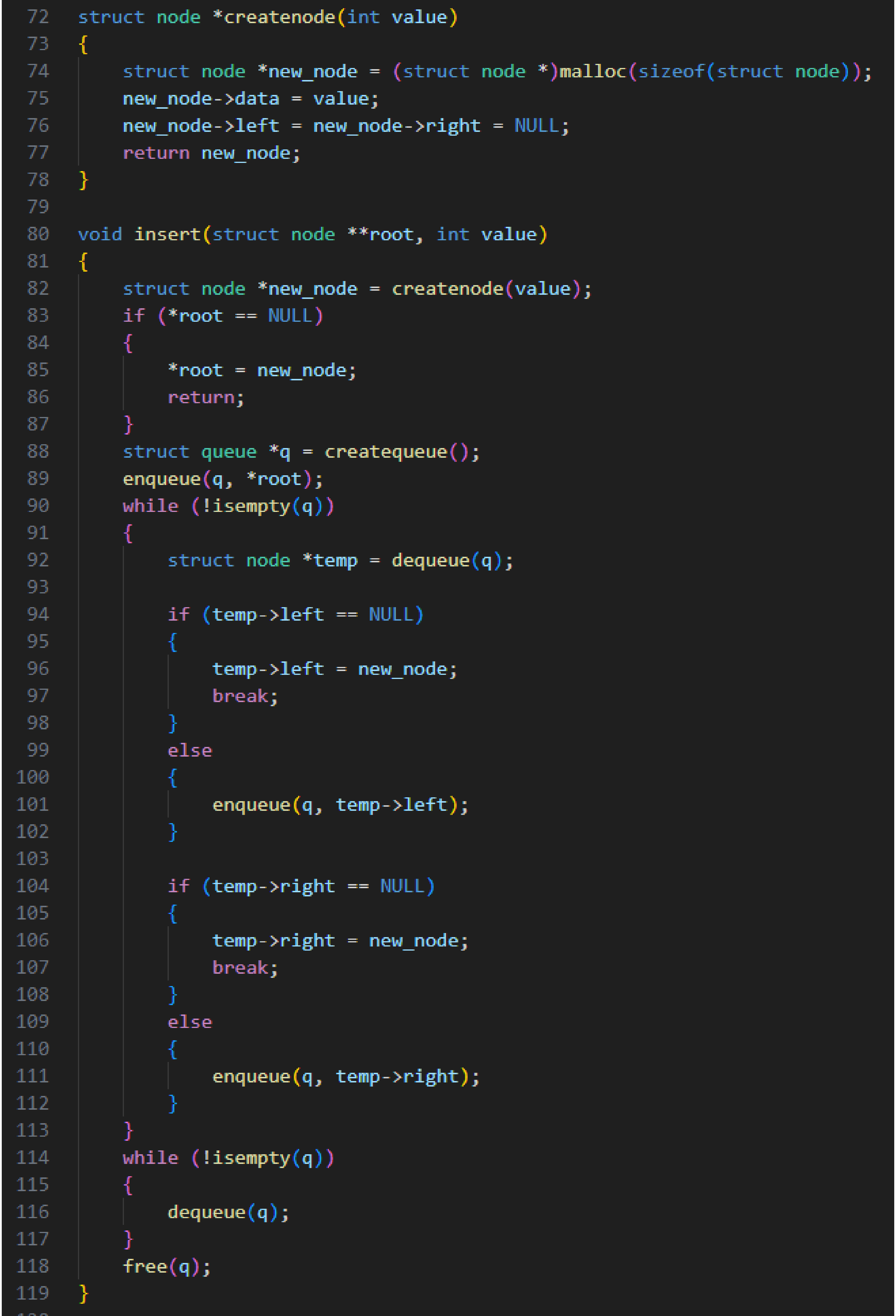


SEM-2 > DSA-ASS-12 > 12_2_bt_inprepost.c > ...

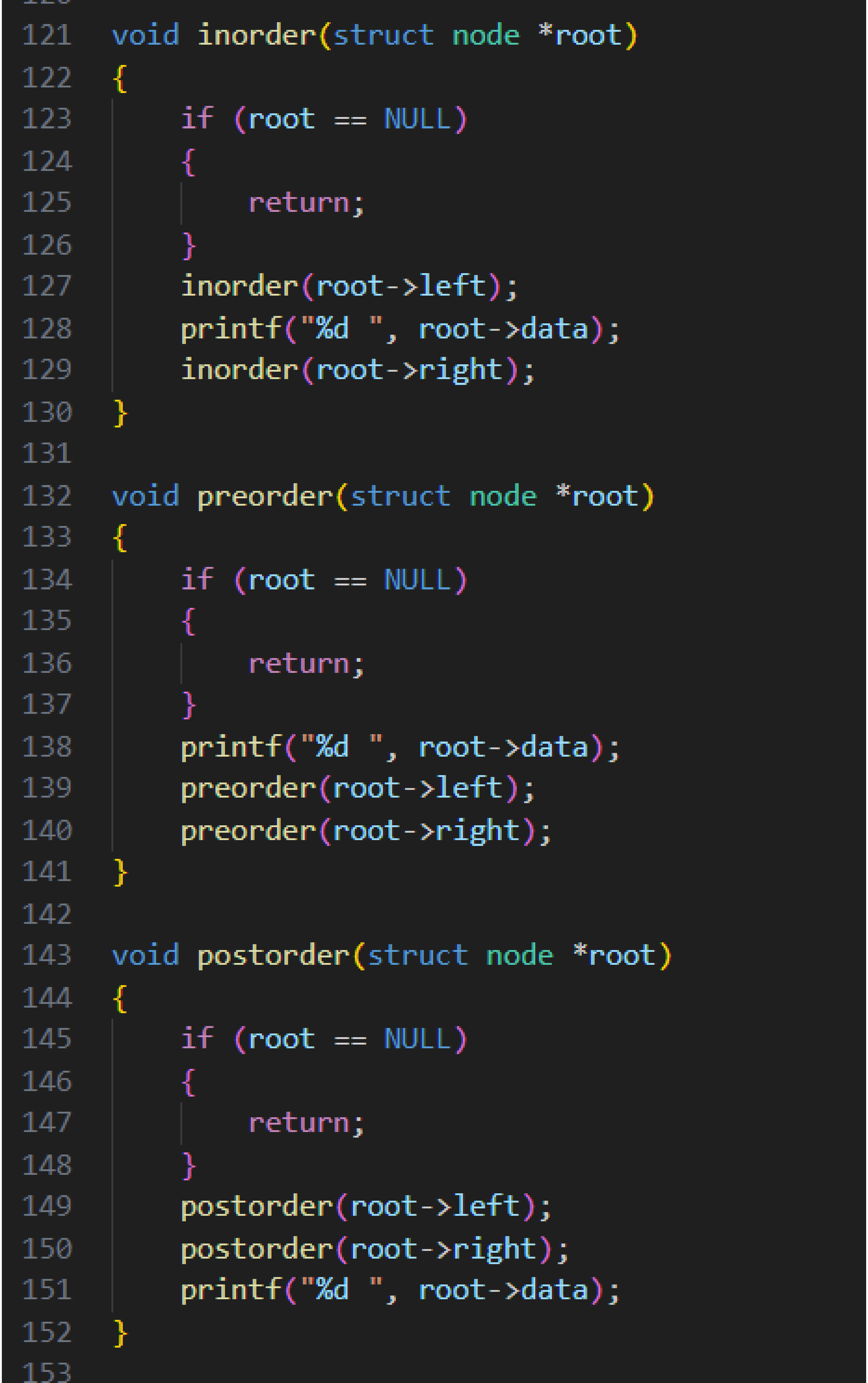
```c
/*  Write a C/C++ program that ask the user to enter 10 integer arr.
Use these arr to construct a binary tree with 10 nodes.
After constructing the tree, perform and display the inorder, preorder, and postorder traversals. */

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

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

struct queuenode
{
    struct node *treenode;
    struct queuenode *next;
};

struct queue
{
    struct queuenode *front;
    struct queuenode *rear;
};

struct queue *createqueue()
{
    struct queue *q = (struct queue *)malloc(sizeof(struct queue));
    q->front = q->rear = NULL;
    return q;
}

```



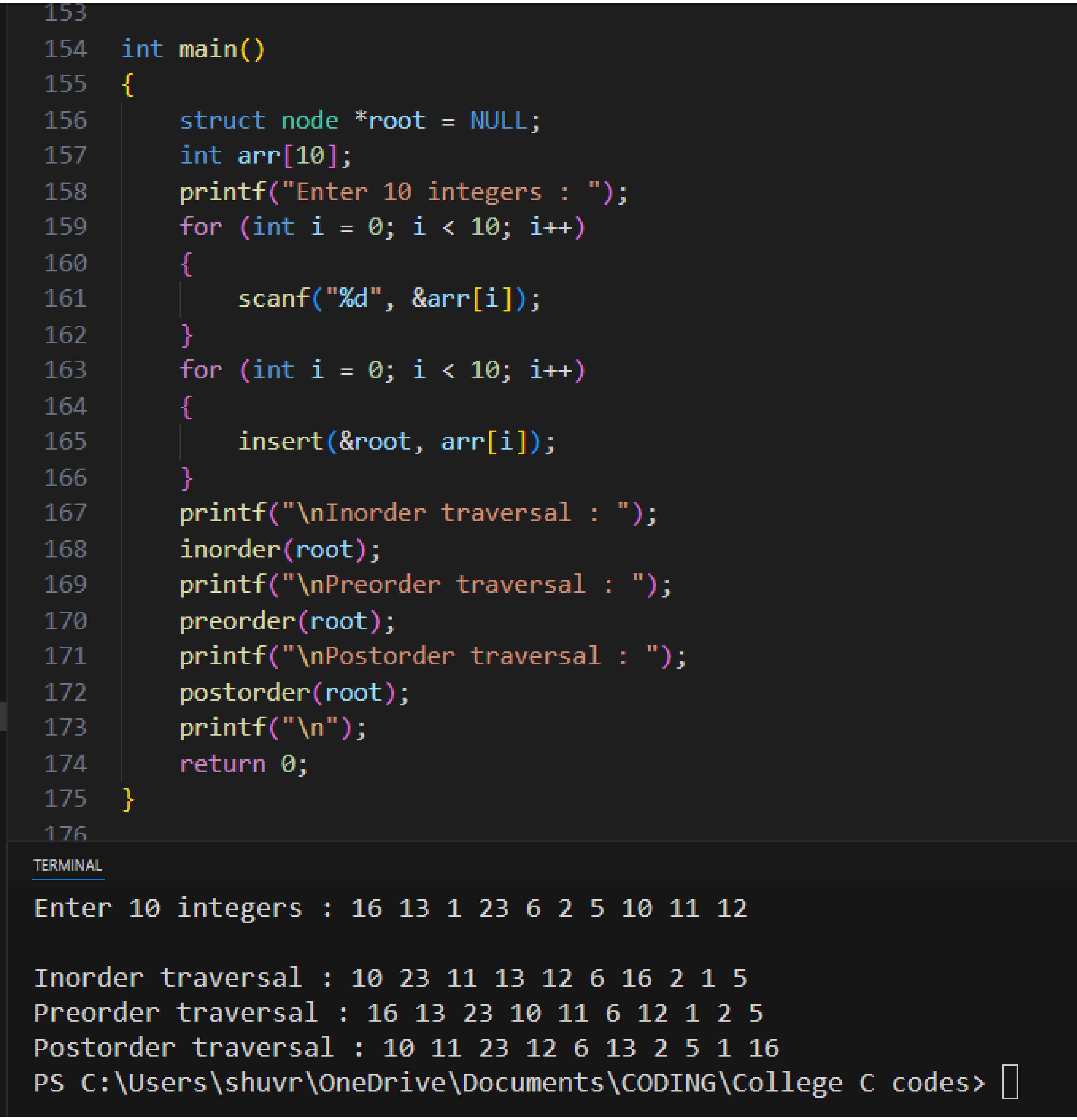
```c
void enqueue(struct queue *q, struct node *node)
{
    struct queuenode *temp = (struct queuenode *)malloc(sizeof(struct queuenode));
    temp->treenode = node;
    temp->next = NULL;
    if (q->rear == NULL)
    {
        q->front = q->rear = temp;
    }
    else
    {
        q->rear->next = temp;
        q->rear = temp;
    }
}

struct node *dequeue(struct queue *q)
{
    if (q->front == NULL)
    {
        return NULL;
    }
    struct queuenode *temp = q->front;
    struct node *node = temp->treenode;
    q->front = q->front->next;
    if (q->front == NULL)
    {
        q->rear = NULL;
    }
    free(temp);
    return node;
}

int isempty(struct queue *q)
{
    return (q->front == NULL);
}
```



```c
struct node *createnode(int value)
{
    struct node *new_node = (struct node *)malloc(sizeof(struct node));
    new_node->data = value;
    new_node->left = new_node->right = NULL;
    return new_node;
}

void insert(struct node **root, int value)
{
    struct node *new_node = createnode(value);
    if (*root == NULL)
    {
        *root = new_node;
        return;
    }
    struct queue *q = createqueue();
    enqueue(q, *root);
    while (!isempty(q))
    {
        struct node *temp = dequeue(q);

        if (temp->left == NULL)
        {
            temp->left = new_node;
            break;
        }
        else
        {
            enqueue(q, temp->left);
        }

        if (temp->right == NULL)
        {
            temp->right = new_node;
            break;
        }
        else
        {
            enqueue(q, temp->right);
        }
    }
    while (!isempty(q))
    {
        dequeue(q);
    }
    free(q);
}
```

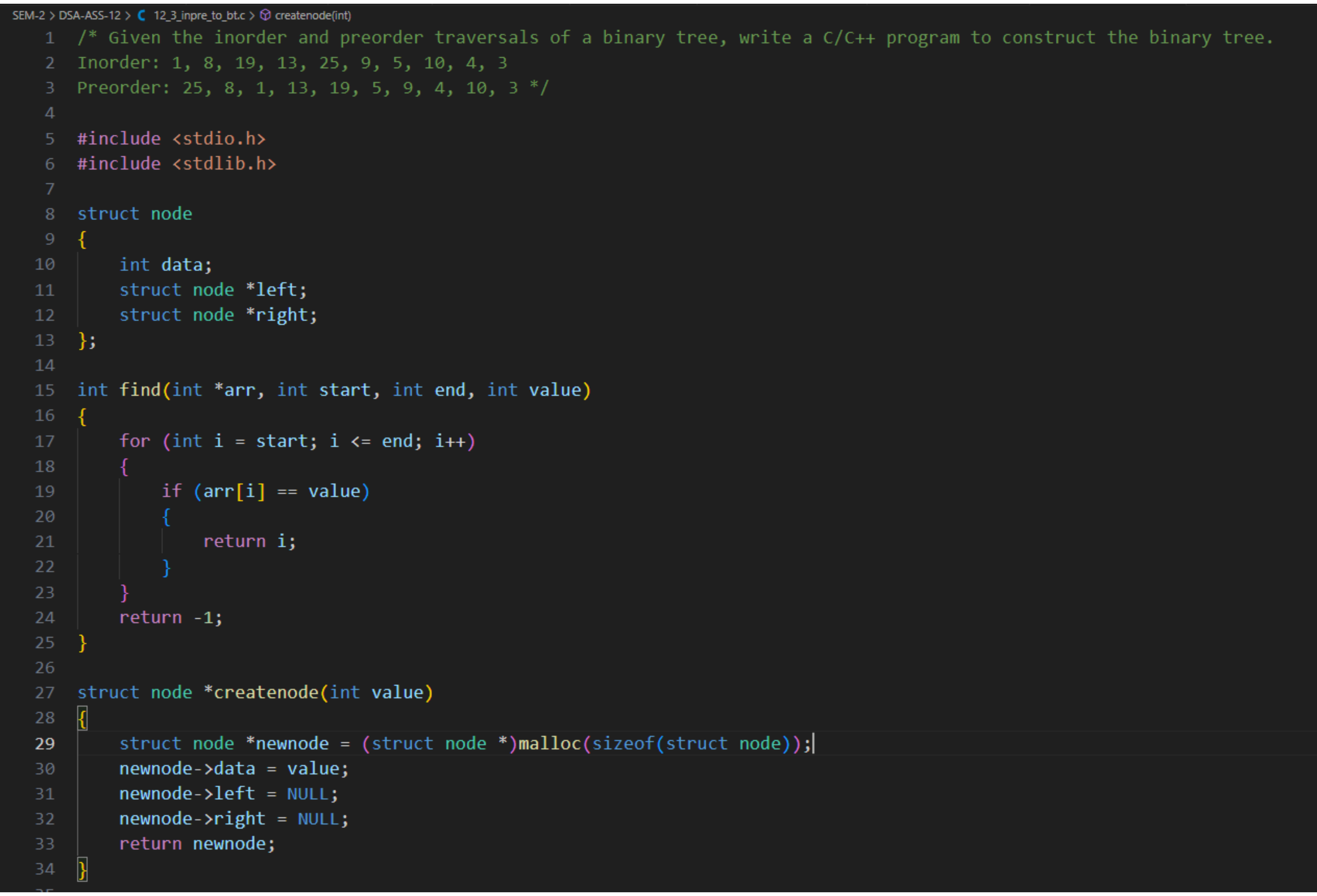


```c
void inorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    inorder(root->left);
    printf("%d ", root->data);
    inorder(root->right);
}

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

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

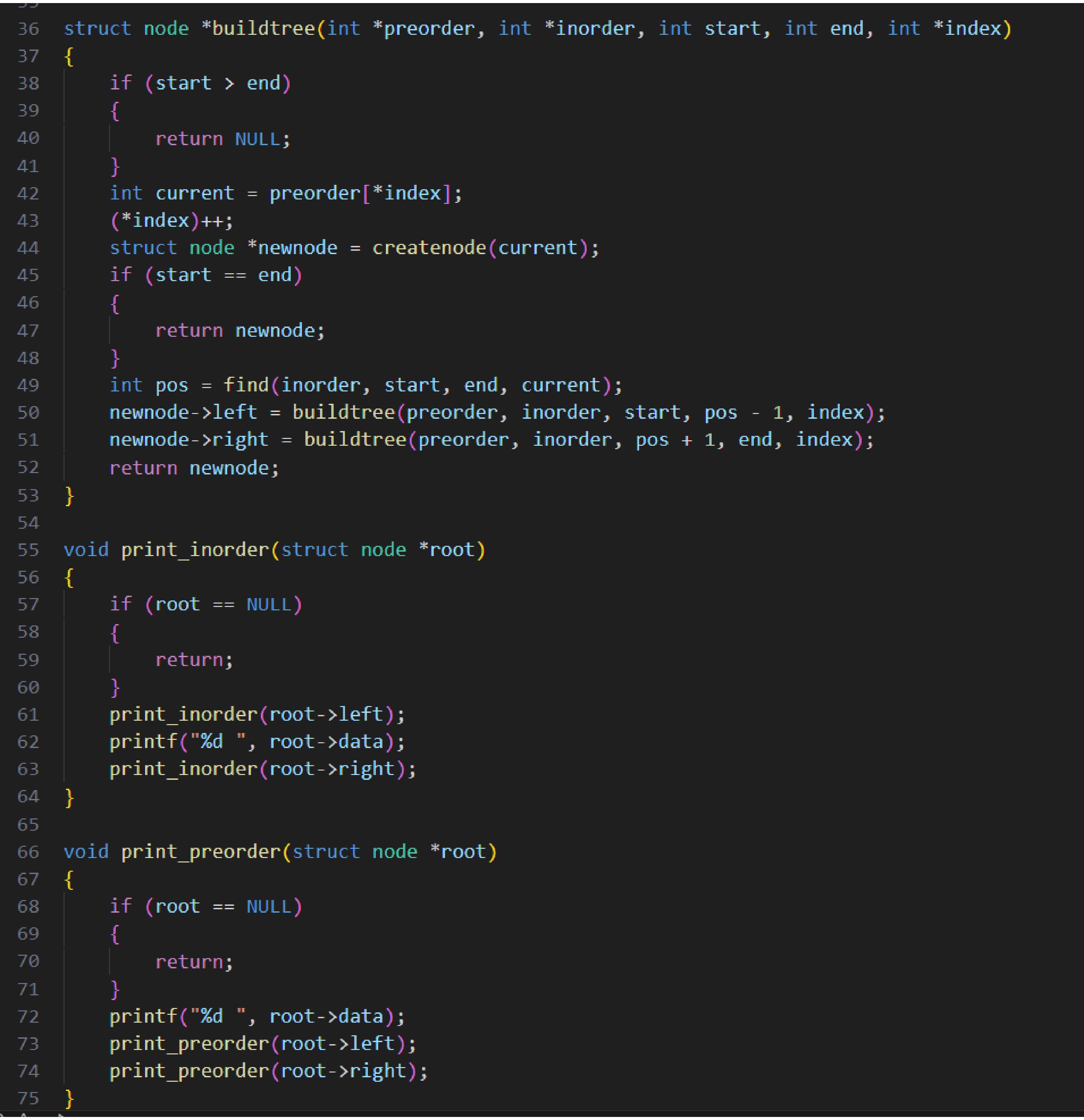


```c
int main()
{
    struct node *root = NULL;
    int arr[10];
    printf("Enter 10 integers : ");
    for (int i = 0; i < 10; i++)
    {
        scanf("%d", &arr[i]);
    }
    for (int i = 0; i < 10; i++)
    {
        insert(&root, arr[i]);
    }
    printf("\nInorder traversal : ");
    inorder(root);
    printf("\nPreorder traversal : ");
    preorder(root);
    printf("\nPostorder traversal : ");
    postorder(root);
    printf("\n");
    return 0;
}
```

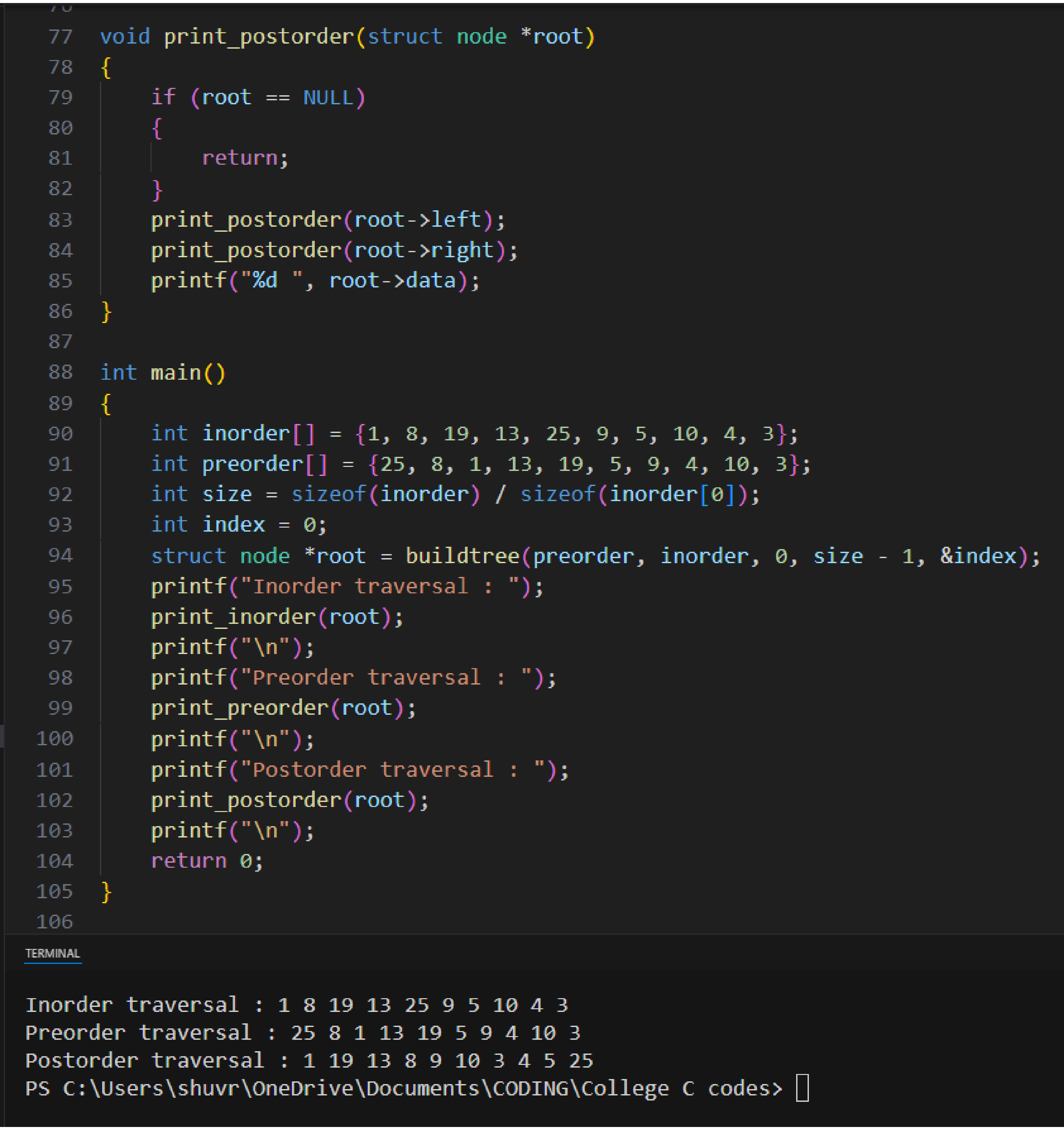
```
Enter 10 integers : 16 13 1 23 6 2 5 10 11 12

Inorder traversal : 10 23 11 13 12 6 16 2 1 5
Preorder traversal : 16 13 23 10 11 6 12 1 2 5
Postorder traversal : 10 11 23 12 6 13 2 5 1 16
PS C:\Users\shuvr\OneDrive\Documents\CODING\College C codes> 
```



```c
/* Given the inorder and preorder traversals of a binary tree, write a C/C++ program to construct the binary tree.
Inorder: 1, 8, 19, 13, 25, 9, 5, 10, 4, 3
Preorder: 25, 8, 1, 13, 19, 5, 9, 4, 10, 3 */

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

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

int find(int *arr, int start, int end, int value)
{
    for (int i = start; i <= end; i++)
    {
        if (arr[i] == value)
        {
            return i;
        }
    }
    return -1;
}

struct node *createnode(int value)
{
    struct node *newnode = (struct node *)malloc(sizeof(struct node));
    newnode->data = value;
    newnode->left = NULL;
    newnode->right = NULL;
    return newnode;
}
```



```c
struct node *buildtree(int *preorder, int *inorder, int start, int end, int *index)
{
    if (start > end)
    {
        return NULL;
    }
    int current = preorder[*index];
    (*index)++;
    struct node *newnode = createnode(current);
    if (start == end)
    {
        return newnode;
    }
    int pos = find(inorder, start, end, current);
    newnode->left = buildtree(preorder, inorder, start, pos - 1, index);
    newnode->right = buildtree(preorder, inorder, pos + 1, end, index);
    return newnode;
}

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

void print_preorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    printf("%d ", root->data);
    print_preorder(root->left);
    print_preorder(root->right);
}
```



```c
void print_postorder(struct node *root)
{
    if (root == NULL)
    {
        return;
    }
    print_postorder(root->left);
    print_postorder(root->right);
    printf("%d ", root->data);
}

int main()
{
    int inorder[] = {1, 8, 19, 13, 25, 9, 5, 10, 4, 3};
    int preorder[] = {25, 8, 1, 13, 19, 5, 9, 4, 10, 3};
    int size = sizeof(inorder) / sizeof(inorder[0]);
    int index = 0;
    struct node *root = buildtree(preorder, inorder, 0, size - 1, &index);
    printf("Inorder traversal : ");
    print_inorder(root);
    printf("\n");
    printf("Preorder traversal : ");
    print_preorder(root);
    printf("\n");
    printf("Postorder traversal : ");
    print_postorder(root);
    printf("\n");
    return 0;
}
```

```
Inorder traversal : 1 8 19 13 25 9 5 10 4 3
Preorder traversal : 25 8 1 13 19 5 9 4 10 3
Postorder traversal : 1 19 13 8 9 10 3 4 5 25
PS C:\Users\shuvr\OneDrive\Documents\CODING\College C codes> 
```