```c
#include <stdio.h>
//https://www.codesdope.com/blog/article/binary-trees-in-c-array-representation-and-travers/
/*
           D
          / \
         /   \
        /     \
       A       F
      / \     / \
     /   \   /   \
    E     B R     T
   / \     /     / \
  G   Q   V     J   L
*/

// variable to store maximum number of nodes
int complete_node = 15;

// array to store the tree
char tree[] = {'\0', 'D', 'A', 'F', 'E', 'B', 'R', 'T', 'G', 'Q', '\0', '\0', 'V', '\0', 'J', 'L'};

int get_right_child(int index)
{
    // node is not null
    // and the result must lie within the number of nodes for a complete binary tree
    if (tree[index] != '\0' && ((2 * index) + 1) <= complete_node)
        return (2 * index) + 1;
    // right child doesn't exist
    return -1;
}

int get_left_child(int index)
{
    // node is not null
    // and the result must lie within the number of nodes for a complete binary tree
    if (tree[index] != '\0' && (2 * index) <= complete_node)
        return 2 * index;
    // left child doesn't exist
    return -1;
}

void preorder(int index)
{
    // checking for valid index and null node
    if (index > 0 && tree[index] != '\0')
    {
        printf(" %c ", tree[index]);      // visiting root
```

```
        preorder(get_left_child(index));  //visiting left subtree
        preorder(get_right_child(index)); //visiting right subtree
    }
}

void postorder(int index)
{
    // checking for valid index and null node
    if (index > 0 && tree[index] != '\0')
    {
        postorder(get_left_child(index));  //visiting left subtree
        postorder(get_right_child(index)); //visiting right subtree
        printf(" %c ", tree[index]);       //visiting root
    }
}

void inorder(int index)
{
    // checking for valid index and null node
    if (index > 0 && tree[index] != '\0')
    {
        inorder(get_left_child(index));  //visiting left subtree
        printf(" %c ", tree[index]);     //visiting root
        inorder(get_right_child(index)); // visiting right subtree
    }
}

int main()
{
    printf("Preorder:\n");
    preorder(1);
    printf("\nPostorder:\n");
    postorder(1);
    printf("\nInorder:\n");
    inorder(1);
    printf("\n");
    return 0;
}
```