

## PrePostorder Traversal in C++

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;

// Node structure definition
struct Node {
    int data;
    vector<Node*> children;
};

// Function to display the tree structure
void display(Node* node) {
    cout << node->data << " -> ";
    for (Node* child : node->children) {
        cout << child->data << ", ";
    }
    cout << "." << endl;

    for (Node* child : node->children) {
        display(child);
    }
}

// Function to construct the tree from an array
Node* construct(vector<int>& arr) {
    Node* root = nullptr;
    vector<Node*> st;

    for (int i = 0; i < arr.size(); ++i) {
        if (arr[i] == -1) {
            st.pop_back();
        } else {
            Node* t = new Node();
            t->data = arr[i];

            if (!st.empty()) {
                st.back()->children.push_back(t);
            } else {
                root = t;
            }

            st.push_back(t);
        }
    }

    return root;
}

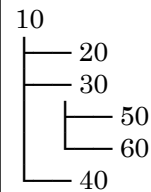
// Function to perform pre-order, post-order, and edge
printing traversals
void traversals(Node* node) {
    // Print Node Pre
    cout << "Node Pre " << node->data << endl;

    // Print Edge Pre
    for (Node* child : node->children) {
        cout << "Edge Pre " << node->data << "--" <<
child->data << endl;
        traversals(child);
    }
    cout << "Edge Post " << node->data << "--" <<
```

**Input Array:**

**{10, 20, -1, 30, 50, -1, 60, -1, -1, 40, -1, -1}**

✔ **Constructed Tree:**



🔄 Dry Run Table for traversals()

Step	Current Node	Action Type	Output
1	10	Node Pre	Node Pre 10
2	10 → 20	Edge Pre	Edge Pre 10--20
3	20	Node Pre	Node Pre 20
4	20	Node Post	Node Post 20
5	10 ← 20	Edge Post	Edge Post 10--20
6	10 → 30	Edge Pre	Edge Pre 10--30
7	30	Node Pre	Node Pre 30
8	30 → 50	Edge Pre	Edge Pre 30--50
9	50	Node Pre	Node Pre 50
10	50	Node Post	Node Post 50
11	30 ← 50	Edge Post	Edge Post 30--50
12	30 → 60	Edge Pre	Edge Pre 30--60
13	60	Node Pre	Node Pre 60
14	60	Node Post	Node Post 60
15	30 ← 60	Edge Post	Edge Post 30--60
16	30	Node Post	Node Post 30
17	10 ← 30	Edge Post	Edge Post 10--30
18	10 → 40	Edge Pre	Edge Pre 10--40
19	40	Node Pre	Node Pre 40
20	40	Node Post	Node Post 40
21	10 ← 40	Edge Post	Edge Post 10--40

<pre>child-&gt;data &lt;&lt; endl; }  // Print Node Post cout &lt;&lt; "Node Post " &lt;&lt; node-&gt;data &lt;&lt; endl; }  int main() {     vector&lt;int&gt; arr = {10, 20, -1, 30, 50, -1, 60, -1, -1, 40, -1, -1};      Node* root = construct(arr);      // Perform pre-order, post-order, and edge printing traversals     traversals(root);      // Clean up memory (not necessary in this simple example but good practice)     // You would typically have a function to delete the tree     return 0; }</pre>	<table><tr><td></td><td></td><td></td><td>-40</td></tr><tr><td>22</td><td>10</td><td>Node Post</td><td>Node Post 10</td></tr></table> <p><b>Final Output (as it would appear on console):</b></p> <pre>Node Pre 10 Edge Pre 10--20 Node Pre 20 Node Post 20 Edge Post 10--20 Edge Pre 10--30 Node Pre 30 Edge Pre 30--50 Node Pre 50 Node Post 50 Edge Post 30--50 Edge Pre 30--60 Node Pre 60 Node Post 60 Edge Post 30--60 Node Post 30 Edge Post 10--30 Edge Pre 10--40 Node Pre 40 Node Post 40 Edge Post 10--40 Node Post 10</pre>				-40	22	10	Node Post	Node Post 10
			-40						
22	10	Node Post	Node Post 10						
<pre>Node Pre 10 Edge Pre 10--20 Node Pre 20 Node Post 20 Edge Post 10--20 Edge Pre 10--30 Node Pre 30 Edge Pre 30--50 Node Pre 50 Node Post 50 Edge Post 30--50 Edge Pre 30--60 Node Pre 60 Node Post 60 Edge Post 30--60 Node Post 30 Edge Post 10--30 Edge Pre 10--40 Node Pre 40 Node Post 40 Edge Post 10--40 Node Post 10</pre>									