

Is Symmetric in C++	
1	<code>std::set</code>
2	<code>std::map</code>
3	<code>std::multiset</code>
4	<code>std::multimap</code>
5	<code>std::unordered_set</code>
6	<code>std::unordered_map</code>
7	<code>std::unordered_multiset</code>
8	<code>std::unordered_multimap</code>
9	<code>std::set</code> (with custom comparator)
10	<code>std::map</code> (with custom comparator)
11	<code>std::multiset</code> (with custom comparator)
12	<code>std::multimap</code> (with custom comparator)
13	<code>std::unordered_set</code> (with custom hash and equality)
14	<code>std::unordered_map</code> (with custom hash and equality)
15	<code>std::unordered_multiset</code> (with custom hash and equality)
16	<code>std::unordered_multimap</code> (with custom hash and equality)

```
#include <iostream>
#include <vector>
#include <stack>
```

```
using namespace std;
```

```
// Node class definition
```

```
class Node {
public:
```

```
int data;
vector<Node*> children;
```

```
vector<Node*> children;
```

```
Node(int val) {
```

```
data = val;
```

$$\left\{ \begin{array}{l} \\ \end{array} \right\}$$

"E... .."

```
// Function to construct the tree
from the given array
```

```
Node* construct(vector<int>&
arr) {
```

```
Node* root = nullptr;
stack<Node*> st;
```

```
stack ~ Node - st,
for (i in 0:n-1) {
  i = i + 1;
}
```

```
for (int i = 0; i < arr.size(); ++i) {
    if (arr[i] == -1) {
```

```

        st.pop();
    } else {

```

```
Node* t = new
Node(arr[i]:
```

```
Node(arr[i]),
    if (i % 2 == 0) {
```

```

if (!st.empty()) {
    st.top()-

```

```
>children.push_back(t);
    } else {
```

```

    root = t;
}

```

3. (i)  $\frac{1}{2} \frac{d}{dt} \int_{\mathbb{R}^n} |\nabla u|^2 dx = \int_{\mathbb{R}^n} u \Delta u dx = - \int_{\mathbb{R}^n} |\nabla u|^2 dx$

```

    st.push(t);
}

```

}	
---	--

```

    return root;
}

```

“Faintly, the light of day is breaking.”

```
// Function to check if two trees
are mirrors of each other
```

```
bool areMirror(Node* n1, Node*
n2) {
```

```

    if (n1->children.size() != n2-
        >children.size()) {

```

```

    >children.size()) {
        return false;
    }
}

```

$\left\{ \begin{array}{l} \text{ } \end{array} \right\}$

```
for (int i = 0; i < n1-
>children.size(): ++i) {
```

```

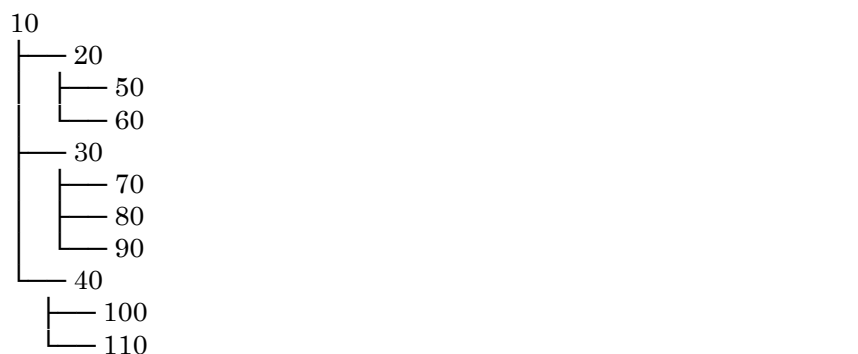
        int j = n1->children.size() - 1
    }
}

```

```
Node* c1 = n1->children[i];
```

```
Node* c2 = n2->children[j];
```

\_\_\_\_\_



### Tabular Dry Run of are Mirror (node1, node2)

Step	node1->data	node2->data	Children Count Match	Comparing Child Pair	Recursive Call	Result
1	10	10	✔ Yes (3 children)	Compare 20 & 40	areMirror(20, 40)	proceeds
2	20	40	✔ Yes (2 children)	Compare 50 & 110	areMirror(50, 110)	✔ true
3	50	110	✔ Yes (0 children)	-	leaf nodes	✔ true
4	20	40	-	Compare 60 & 100	areMirror(60, 100)	✔ true
5	60	100	✔ Yes (0 children)	-	leaf nodes	✔ true
6	20 & 40	done	All children matched	-	return to previous	✔ true
7	10	10	-	Compare 30 & 30 (middle node)	areMirror(30, 30)	proceeds
8	30	30	✔ Yes (3 children)	Compare 70 & 90	areMirror(70, 90)	✔ true
9	70	90	✔ Yes (0	-	leaf nodes	✔ true

<pre>         if (!areMirror(c1, c2)) {             return false;         }     }      return true; }  // Function to check if a tree is symmetric bool IsSymmetric(Node* node) {     return areMirror(node, node); }  // Main function int main() {     vector&lt;int&gt; arr = {10, 20, 50, -1, 60, -1, -1, 30, 70, -1, 80, -1, 90, -1, -1, 40, 100, -1, 110, -1, -1, -1};      Node* root = construct(arr);     bool sym = IsSymmetric(root);     cout &lt;&lt; boolalpha &lt;&lt; sym &lt;&lt; endl;      return 0; } </pre>				children)			
	10	30	30	-	Compare 80 & 80	areMirror(80, 80)	✔ true
	11	80	80	✔ Yes (0 children)	-	leaf nodes	✔ true
	12	30	30	-	Compare 90 & 70	areMirror(90, 70)	✔ true
	13	90	70	✔ Yes (0 children)	-	leaf nodes	✔ true
	14	30 & 30	done	All children matched	-	return to previous	✔ true
	15	10	10	-	Compare 40 & 20	already compared in step 1	✔ true
	16	10 & 10	done	All pairs matched	-	final result	✔ true
<p>✔ <b>Final Result:</b></p> <p>true</p>							

true