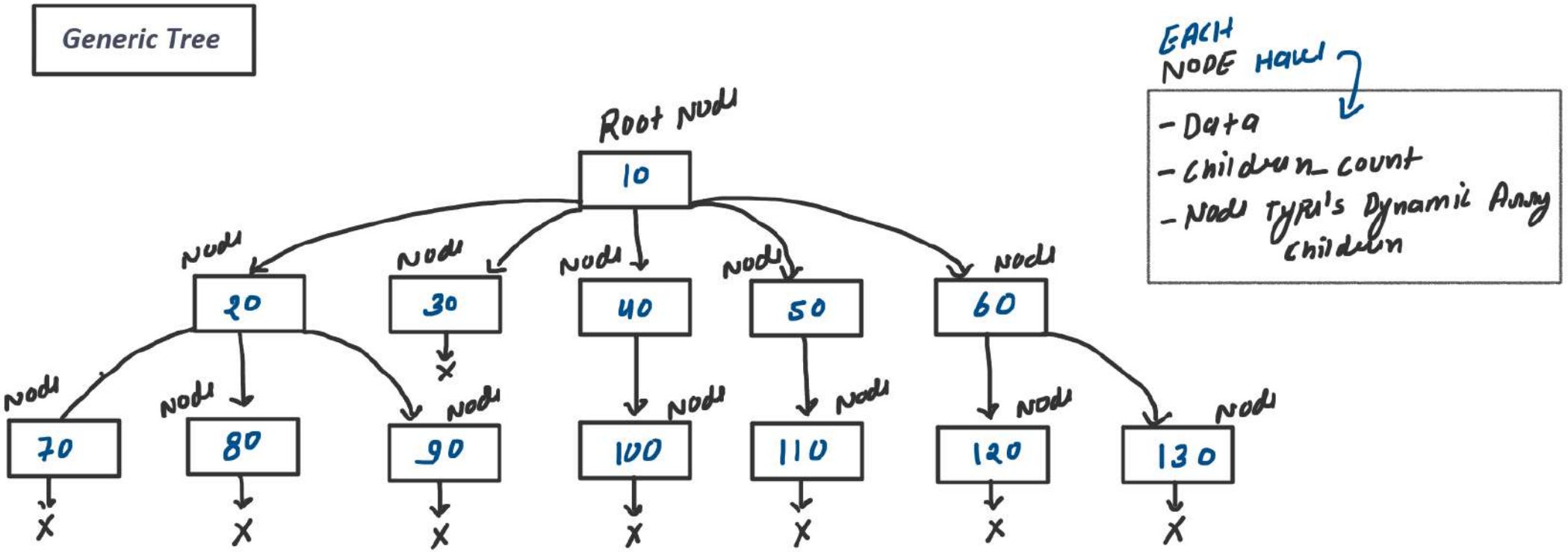
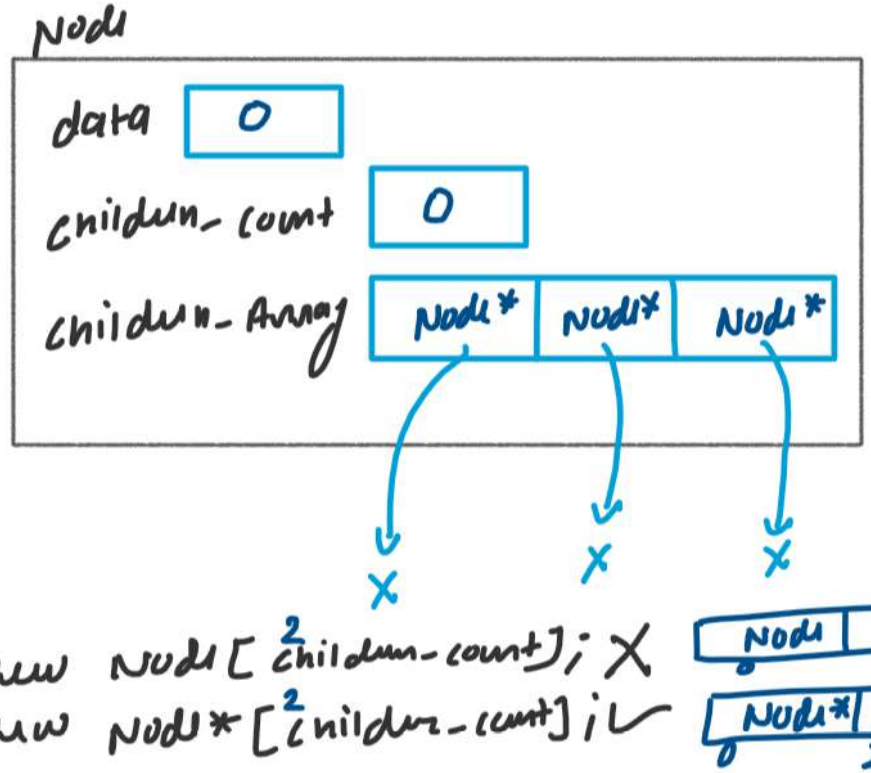

Generic Tree
Root Node
10
Node
20
30
40
50
60
70
80
90
100
110
120
130
EACH NODE HAS
- Data
- children_count
- Node type's Dynamic Array children

```
class Node {

    int data;

    int children_count

    Node* *children;
};
```

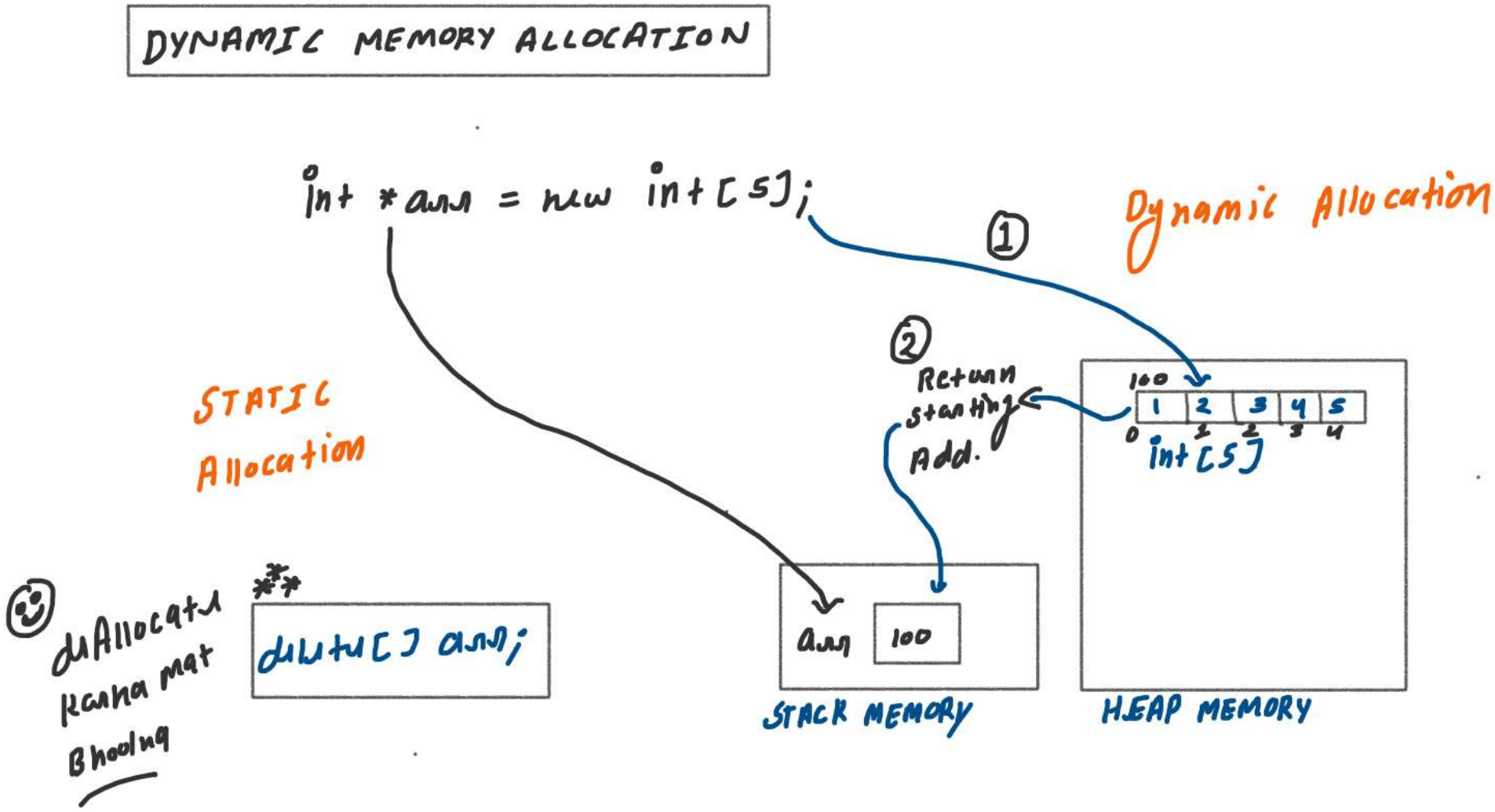
Children store Node* type data

This is a Dynamic Array



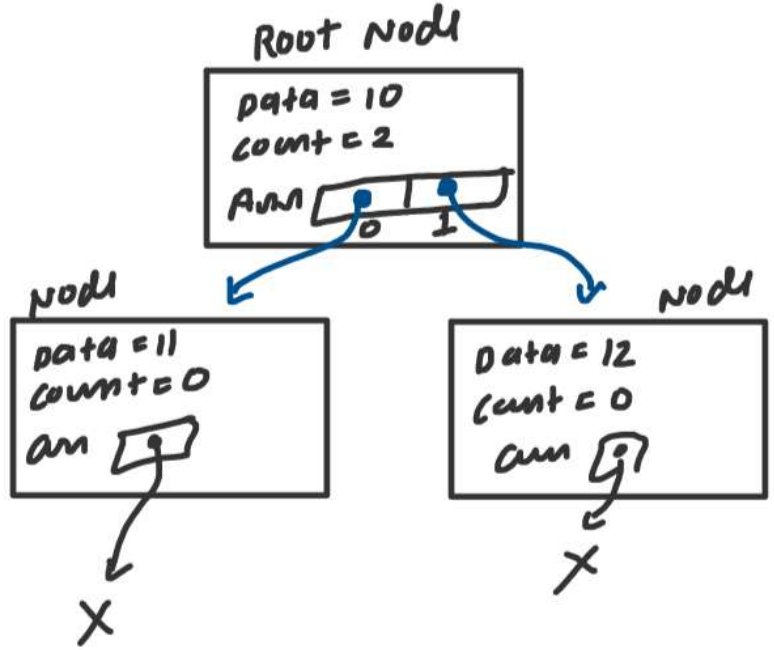

```
X Node* children = new Node[2children_count]; X
Node** children = new Node*[2children_count]; ✓
```

DYNAMIC MEMORY ALLOCATION



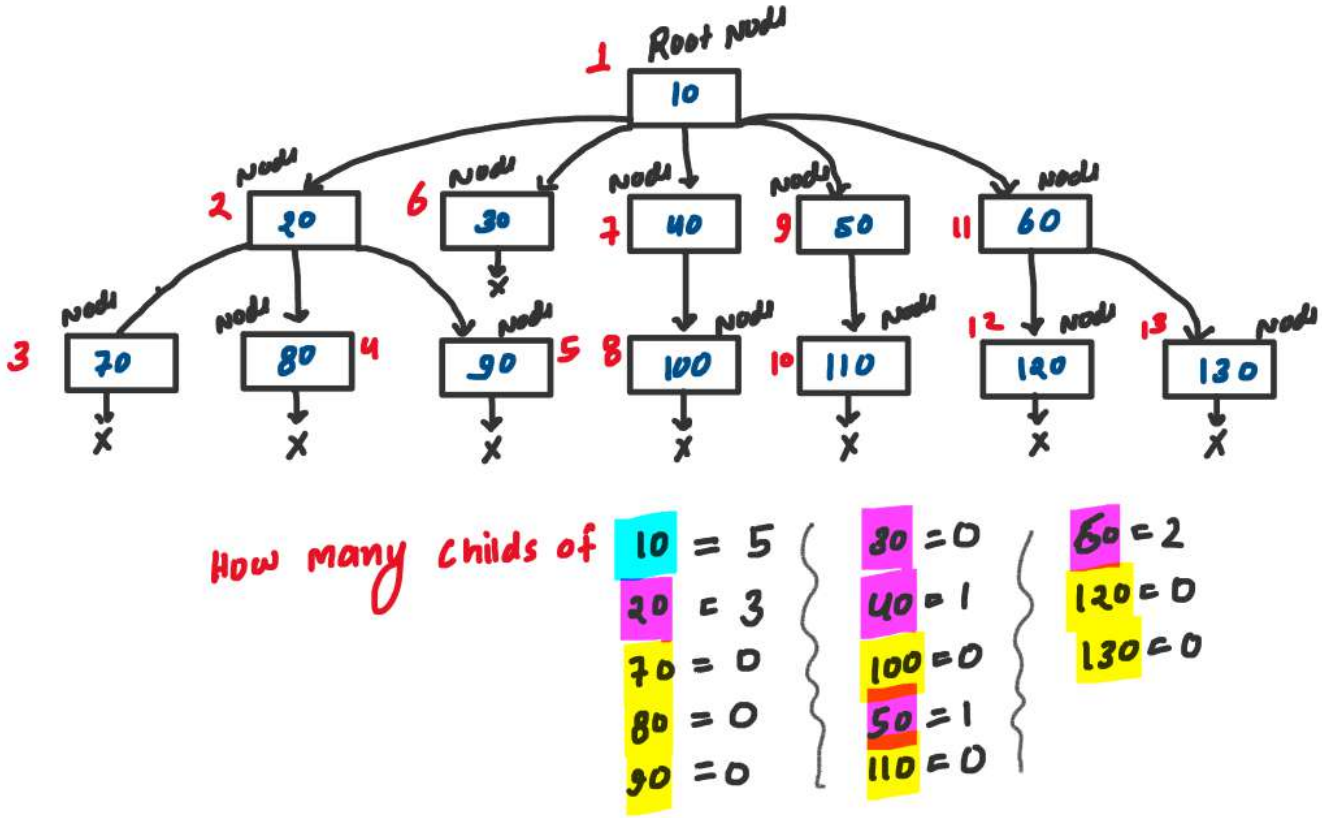
Example 1:

```
TREE INPUT
Enter root data: 10
Enter Children count for 10 node: 2
Enter root data: 11
Enter Children count for 11 node: 0
Enter root data: 12
Enter Children count for 12 node: 0

TREE OUTPUT
10
11 12
```



Example 2:

```
TREE INPUT
Enter root data: 10
Enter Children count for 10 node: 5
Enter root data: 20
Enter Children count for 20 node: 3
Enter root data: 70
Enter Children count for 70 node: 0
Enter root data: 80
Enter Children count for 80 node: 0
Enter root data: 90
Enter Children count for 90 node: 0
Enter root data: 30
Enter Children count for 30 node: 0
Enter root data: 40
Enter Children count for 40 node: 1
Enter root data: 100
Enter Children count for 100 node: 0
Enter root data: 50
Enter Children count for 50 node: 1
Enter root data: 110
Enter Children count for 110 node: 0
Enter root data: 60
Enter Children count for 60 node: 2
Enter root data: 120
Enter Children count for 120 node: 0
Enter root data: 130
Enter Children count for 130 node: 0

TREE OUTPUT
10
20 30 40 50 60
70 80 90 100 110 120 130
```



```cpp
// GENERIC TREE PROBLEM
#include <iostream>
#include<queue>
using namespace std;

class Node{
    public:
    int data;
    int children_count;
    Node** children;

    Node(int value) {
        this->data = value;
        this->children_count = 0;
        this->children = NULL;
    }
};

Node* takeInput(){
    int data, count;
    cout<<"Enter root data: ";
    cin>>data;
    cout<<"Enter Children count for "<<data<<" node: ";
    cin>>count;

    // Create Root Node
    Node* root = new Node(data);

    // Create Child Node of Root Node
    root->children_count = count;
    // Dynamic Array to store links to children
    root->children = new Node[count];
    for(int i=0;i<count;i++){
        root->children[i] = takeInput();
    }
    return root;
}

int main() {
    takeInput();
    return 0;
}
```

```cpp
// LEVEL ORDER TRAVERSAL OF A GENERIC TREE PROBLEM
#include <iostream>
#include<queue>
using namespace std;

class Node{
    ...
};

Node* takeInput(){
    ...
}

void levelOrderPrint(Node* root){
    queue<Node*> q;
    q.push(root);
    q.push(NULL);

    while(!q.empty()){
        auto front = q.front();
        q.pop();
        if(front == NULL){
            cout<<endl;
            if(!q.empty()){
                q.push(NULL);
            }
        }
        else{
            cout<< front->data <<" ";
            for(int i=0;i<front->children_count;i++){
                if(front->children[i]){
                    q.push(front->children[i]);
                }
            }
        }
    }
}

int main() {
    Node* root = takeInput();
    levelOrderPrint(root);
    return 0;
}
```