1. Run the following code, and find the purpose of them. Screenshot the output and explain it.
    a.

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

// Node structure for general tree
struct TreeNode {
    int data;
    vector<TreeNode*> children;  // Can have multiple children
};

// Function to print tree (DFS Traversal)
void printTree(TreeNode* root, int level = 0) {
    if (!root) return;

    // Print with indentation based on level
    for (int i = 0; i < level; i++) cout << "  ";
    cout << root->data << endl;

    for (auto child : root->children)
        printTree(child, level + 1);
}

int main() {
    // Creating tree manually
    TreeNode* root = new TreeNode{1};
    TreeNode* child1 = new TreeNode{2};
    TreeNode* child2 = new TreeNode{3};
    TreeNode* child3 = new TreeNode{4};

    root->children.push_back(child1);
    root->children.push_back(child2);
    child1->children.push_back(child3);

    cout << "Tree Structure:\n";
    printTree(root);

    return 0;
}
```

b. Binary Tree

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

// Node structure for Binary Tree
struct Node {
    int data;
    Node* left;
    Node* right;
};

// Function to create a new node
Node* createNode(int data) {
    Node* newNode = new Node{data, nullptr, nullptr};
    return newNode;
}

// In-order Traversal (Left, Root, Right)
void inorderTraversal(Node* root) {
    if (root) {
        inorderTraversal(root->left);
        cout << root->data << " ";
        inorderTraversal(root->right);
    }
}

int main() {
    // Creating binary tree manually
    Node* root = createNode(10);
    root->left = createNode(5);
    root->right = createNode(15);
    root->left->left = createNode(3);
    root->left->right = createNode(7);
    root->right->left = createNode(12);
    root->right->right = createNode(17);

    cout << "In-order Traversal of Binary Tree:\n";
    inorderTraversal(root);
    cout << endl;

    return 0;
}
```

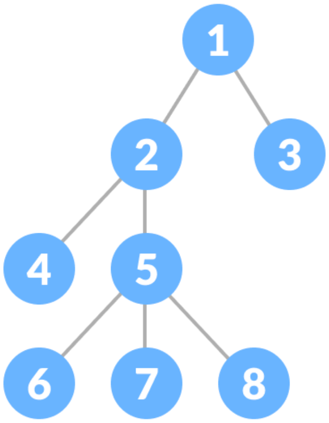
c. Modify the code in 01(a) to find the traversals: inorder, preorder, and postorder.

2. Write the program using binary tree code to perform insertion, deletion, and update. Furthermore, write the code to provide the following sample output:

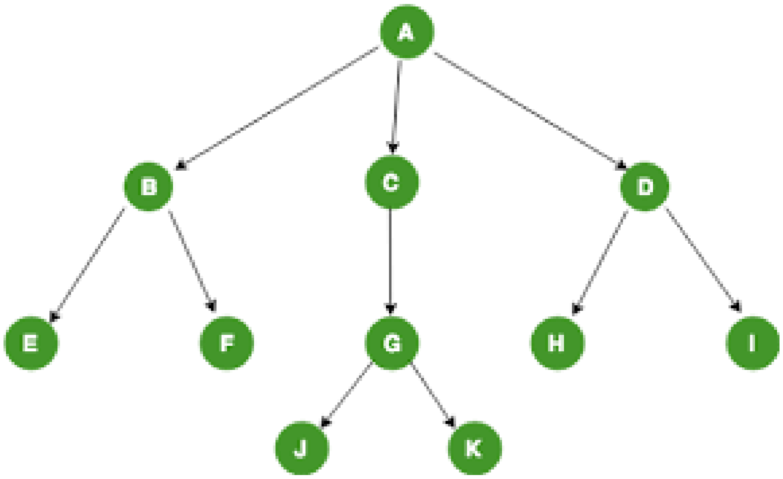
```
In-order Traversal of Tree:
20 30 40 50 60 70 80
After deleting 30:
20 40 50 60 70 80
After updating 70 to 75:
20 40 50 60 75 80
```

3. Create the following tree structure

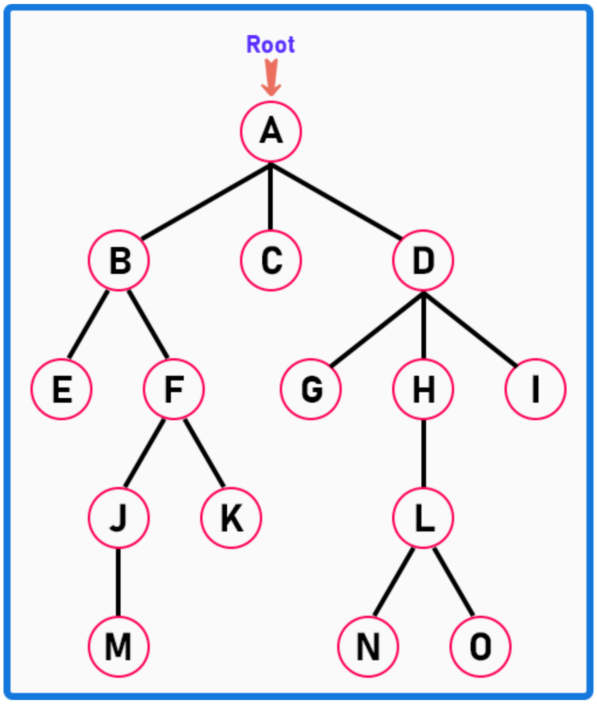
a.



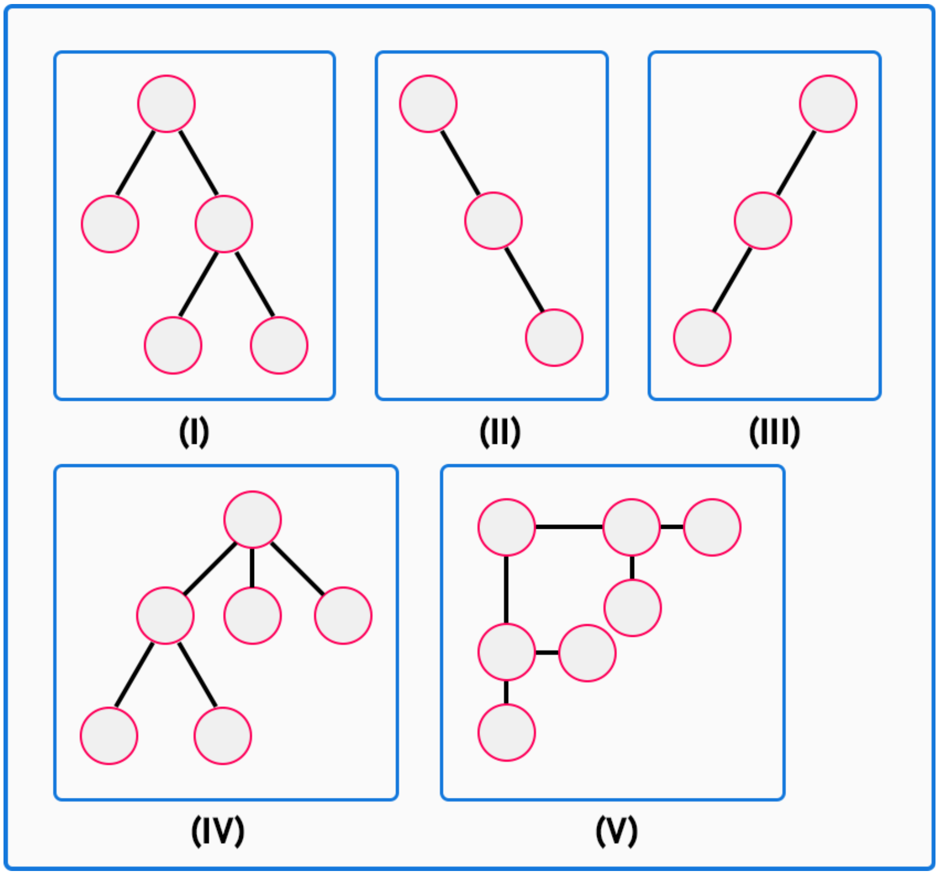
b.



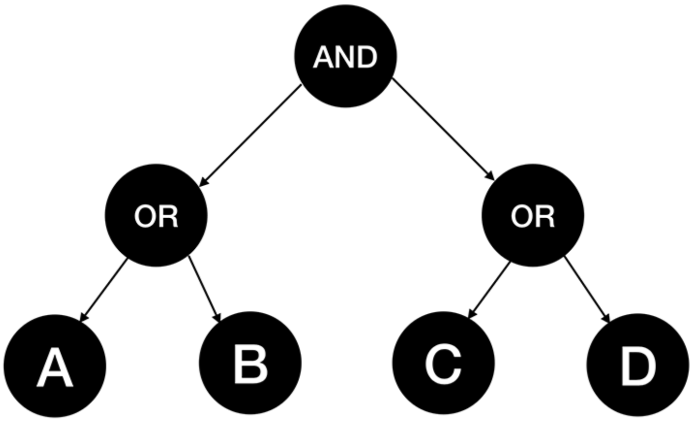
c.



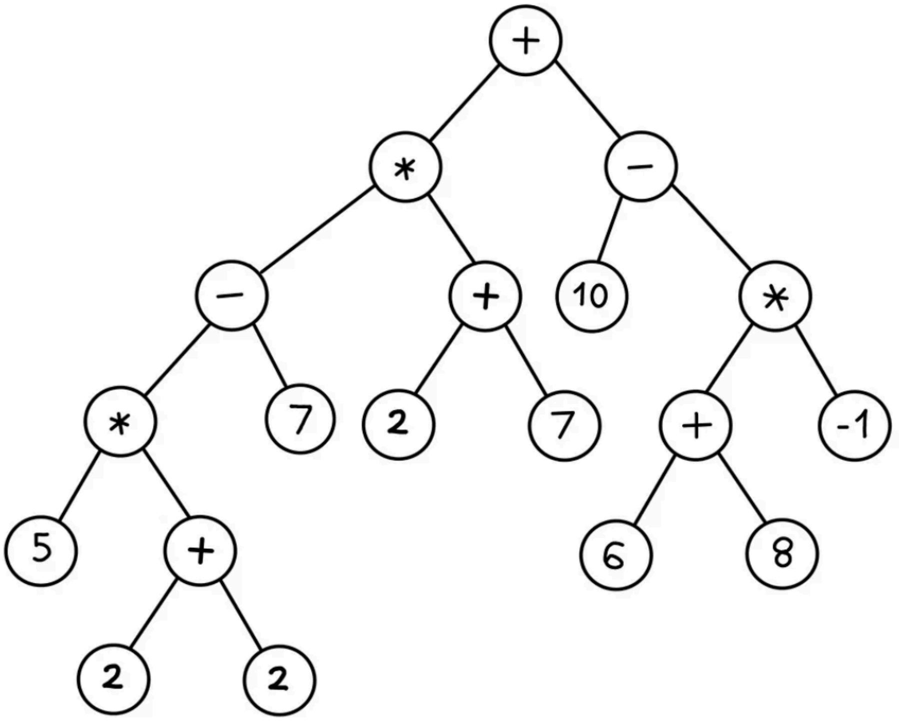
4. Create the following Binary trees and perform the traversals: inorder, preorder and postorder



vi.



vii.



5. You are building an **Employee Management System** for a company where employees are organized in a **hierarchical structure** (Binary Search Tree).

Each Employee Record Includes:

- **Employee ID** (integer) — This will decide position in the tree.
- **Employee Name** (string).
- **Position/Designation** (string) — Example: “Manager”, “Developer”, “HR”, etc.

Tasks:

1. **Insert New Employee:**
    - Insert based on Employee ID (BST rules).
    - Store name & position along with ID.
2. **Display Entire Employee Hierarchy:**
    - Show hierarchy using **In-order**, **Pre-order**, and **Post-order** traversals.
    - Display **ID, Name, and Position** in each node.
3. **Delete an Employee Record:**
    - Remove employee by ID (handle BST deletion cases).
    - Keep hierarchy consistent.
4. **Update Employee Record:**
    - By providing Employee ID, allow user to **update** the:
        - Name
        - Position
        - (Optional: Allow ID update too, but warn about BST rules!)
5. **Retrieve Employee Details by ID:**
    - User enters ID → System searches & displays:

        - Employee Name
        - Position
        - Level in Hierarchy (Optional: Show how deep in the tree they are).

6. **Find Employee with Lowest & Highest ID:**
    - Display their details.
7. **Show Hierarchy Depth (Tree Height):**
    - Indicate the **longest reporting chain**.

SAMPLE MENU

```
1. Add Employee
2. Display Employee Hierarchy
3. Delete Employee
4. Update Employee Details
5. Search Employee by ID
6. Find Lowest & Highest Employee IDs
7. Show Hierarchy Depth
8. Exit
```