Left, Right, top and bottom view of a Binary Tree

Problem: Given a Binary Tree. The task is to print the nodes of the binary tree when viewed from different sides. That is, the left view of the binary tree will contain only those nodes which can be seen when the Binary tree is viewed from left.

Example:

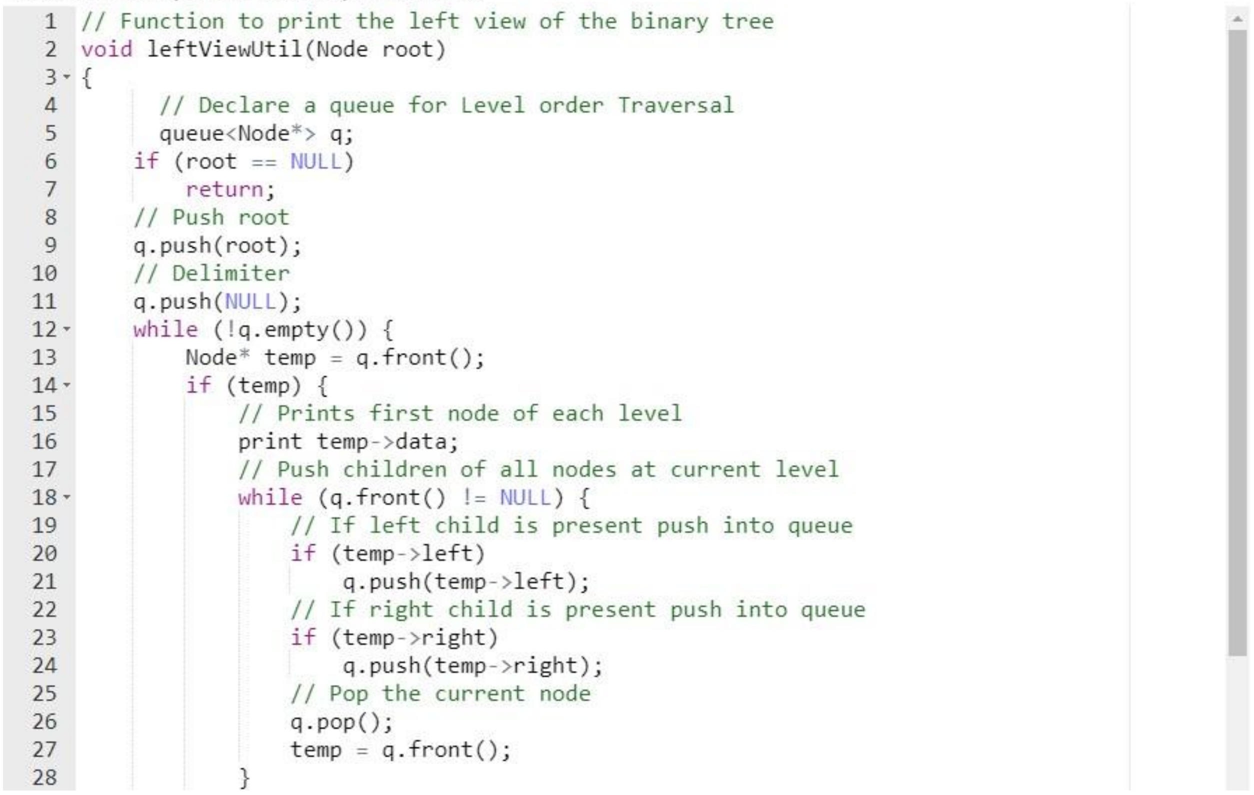
```
Consider the Below Binary Tree:
                              1
                            /   \
                           2     3
                          / \   / \
                         4   5 6   7


Left View of above Tree will be: 1, 2, 4
Right View of above Tree will be: 1, 3, 7
Top View of above Tree will be: 4, 2, 1, 3, 7
Bottom View of above Tree will be: 4, 5, 6, 7
```

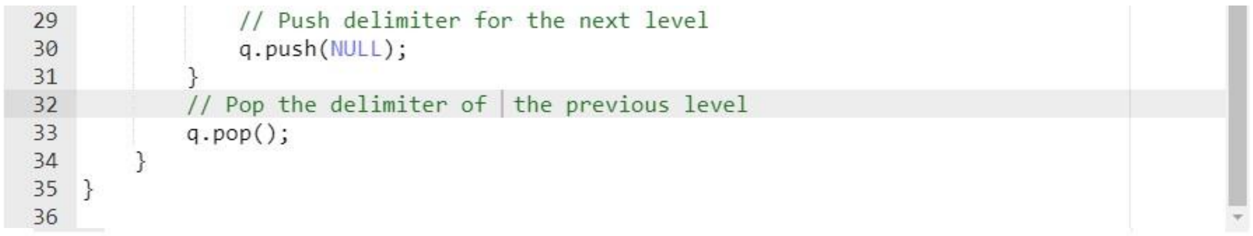
Let us now look at each of the solutions in details:

- **Left View**: A simple solution is to notice that the nodes appearing in the left view of the binary tree are the first nodes at every level. So, the idea is to do a level order traversal of the binary tree using a marker to identify levels and print the first node at every level.

Below is the complete function to print left view:



```
// Function to print the left view of the binary tree
void leftViewUtil(Node root)
{
      // Declare a queue for Level order Traversal
      queue<Node*> q;
    if (root == NULL)
        return;
    // Push root
    q.push(root);
    // Delimiter
    q.push(NULL);
    while (!q.empty()) {
        Node* temp = q.front();
        if (temp) {
            // Prints first node of each level
            print temp->data;
            // Push children of all nodes at current level
            while (q.front() != NULL) {
                // If left child is present push into queue
                if (temp->left)
                    q.push(temp->left);
                // If right child is present push into queue
                if (temp->right)
                    q.push(temp->right);
                // Pop the current node
                q.pop();
                temp = q.front();
            }
```



```
            // Push delimiter for the next level
            q.push(NULL);
        }
        // Pop the delimiter of the previous level
        q.pop();
    }
}
```

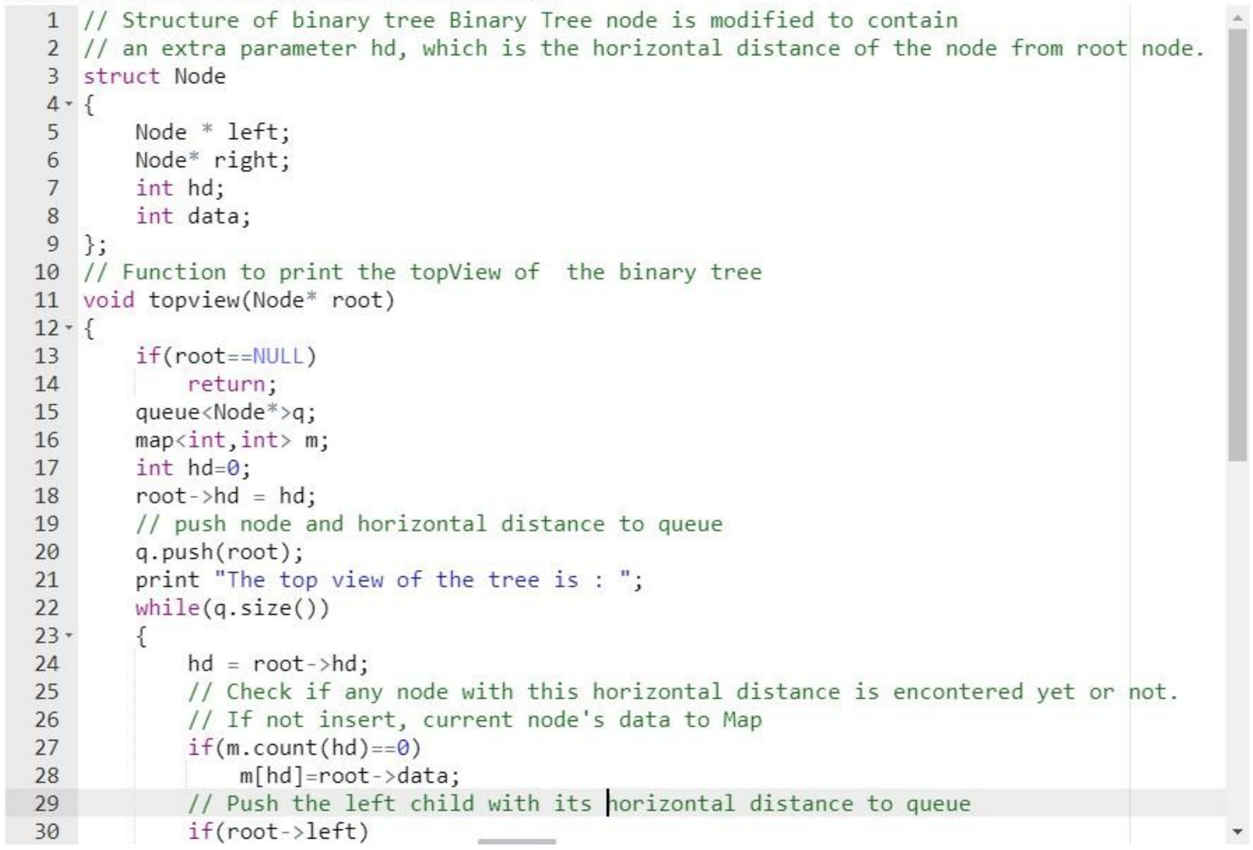
- **Right View**: Printing Right View of the Binary Tree is similar to the above approach of printing the Left view of the tree. The idea is to print the last node present at every level. So, perform a level order traversal using a delimeter to identify levels and print the last node present at every level.

- **Top View**: Top view of a binary tree is the set of nodes visible when the tree is viewed from the top. A node **x** is there in output if x is the topmost node at its horizontal distance. Horizontal distance of left child of a node x is equal to the horizontal distance of x minus 1, and that of right child is the horizontal distance of x plus 1.

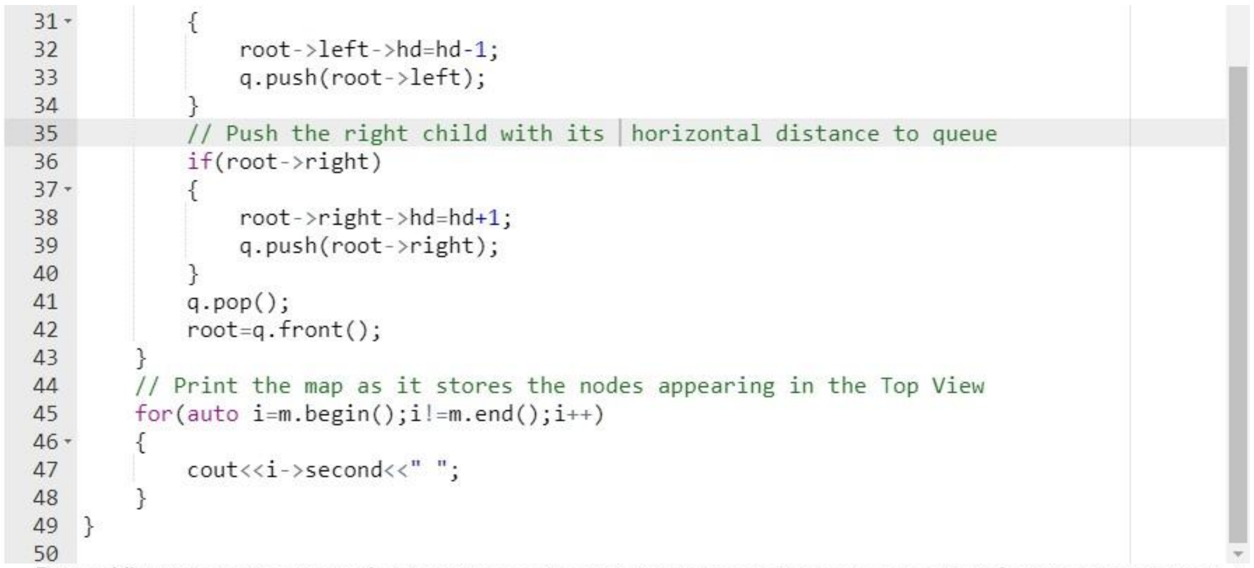
  So, the idea is to do a level order traversal of the tree and calculate the horizontal distance of every node from the root node and print those nodes which are the first nodes of a particular horizontal distance.

  Hashing can be used to keep a check on whether any node with a particular horizontal distance is encountered yet or not.

Below is the function implementing the above approach:



```
// Structure of binary tree Binary Tree node is modified to contain
// an extra parameter hd, which is the horizontal distance of the node from root node.
struct Node
{
    Node * left;
    Node* right;
    int hd;
    int data;
};
// Function to print the topView of  the binary tree
void topview(Node* root)
{
    if(root==NULL)
        return;
    queue<Node*>q;
    map<int,int> m;
    int hd=0;
    root->hd = hd;
    // push node and horizontal distance to queue
    q.push(root);
    print "The top view of the tree is : ";
    while(q.size())
    {
        hd = root->hd;
        // Check if any node with this horizontal distance is encontered yet or not.
        // If not insert, current node's data to Map
        if(m.count(hd)==0)
            m[hd]=root->data;
        // Push the left child with its horizontal distance to queue
        if(root->left)
```



```
        {
            root->left->hd=hd-1;
            q.push(root->left);
        }
        // Push the right child with its horizontal distance to queue
        if(root->right)
        {
            root->right->hd=hd+1;
            q.push(root->right);
        }
        q.pop();
        root=q.front();
    }
    // Print the map as it stores the nodes appearing in the Top View
    for(auto i=m.begin();i!=m.end();i++)
    {
        cout<<i->second<<" ";
    }
}
```

- **Bottom View**: The Bottom View of a binary tree can be printed using the similar approach as that of printing the Top View. A node **x** is there in output if x is the bottom most instead of the top most node at its horizontal distance.

  The process of printing the bottom view is almost the same as that of top view with a little modification that while storing the node's data along with a particular horizontal distance in the map, keep updating the node's data in the map for a particular horizontal distance so that the map contains the last node appearing with a particular horizontal distance instead of first.