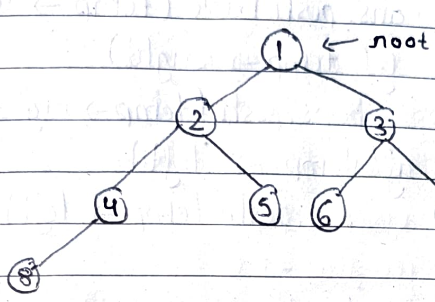
Date ________ Page ________

Day - 156

* Pre-order Traversal Iterative:

⇒



⇒ Preorder: 1 2 4 8 5 3 6 7

⇒ we have to solve without using recursion.

⇒ we know that, recursion uses stack.

⇒ So, here, we have to use stack directly.

⇒ we have to do –
    Print Node
    Left side
    Right side

⇒ So, while solving, first we push root node then push right child then left child.

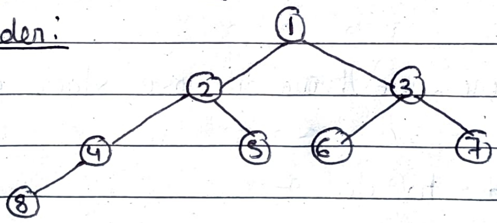
Code

```
vector <int> preorder( Node * root){
        stack< Node *> s;
        s.push (root);
        vector <int> ans;
```

```
    while(!s.empty()){
        Node *temp = s.top();
        s.pop();
        ans.push_back(temp->data);
        if( temp-> right)
            s.push(temp-> right);
        if( temp -> left)
            s.push(temp -> left);
    }
    return ans;
}
```

* **Post Order Traversal:**

Post order:



⇒ 8 4 5 2 6 7 3 1
(L R N)

⇒ We know that we have print nodes in the order — L R N.

⇒ But if we reverse the order — N R L then we can easily solve our problem.

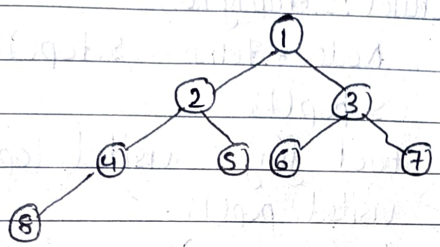
⇒ And the answer, that we will get, we will reverse it that is our required answer.

Date

Page

=> we will use the same previous approach.

* Inorder Traversal:



=> 8 4 2 5 1 6 3 7

=> We have to print that element that we are visiting second time.

=> So, when we get an node and that node is first time, then we are pushing right, node & left.

=> In second time, we will directly print the node.

=> For knowing every node visiting status, we will use an another stack.

=> For every node, we will store their visiting status in the second stack.

Code

```
vector<int> inorder (Node *root){
        stack <Node * > s;
       stack < bool > visited;
```

Date: ____________ Page ________

```
        s.push(root);
        visited.push(0);
        vector<~~ans~~ int> ans;
        while(!s.empty){
            Node * temp = s.top();
            s.pop();
            bool flag = visited.top();
            visited.pop();
            if( flag == 0 ){
                if(temp → right){
                    s.push(temp → right);
                    visited.push(0);
                }
                s.push( temp);
                visited.push( 1);
              if( temp → left){
                    s.push(temp → left)
                    visited.push(0);
              }
          }else{
              ans.push_back(temp→data);
          }
    }
    return ans;
}
```