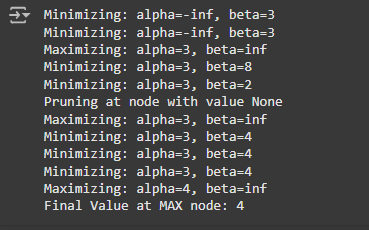
**LAB 10:**

**Program:** Analyse Alpha-Beta pruning method and algorithm and implement the same to write the Alpha-Beta values generated to identify the final value of MAX node and subtrees pruned for a given Game tree.

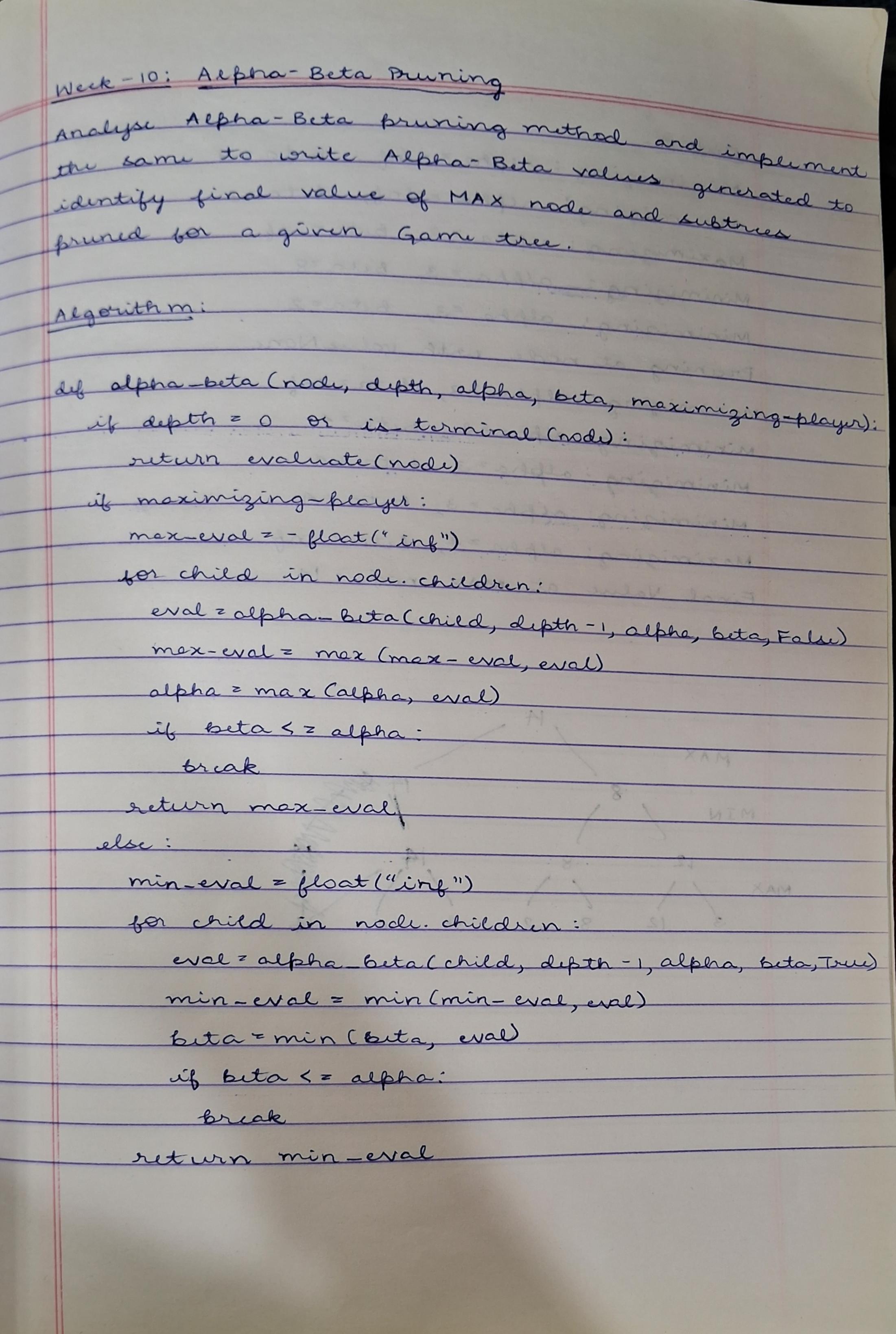
**Code:**

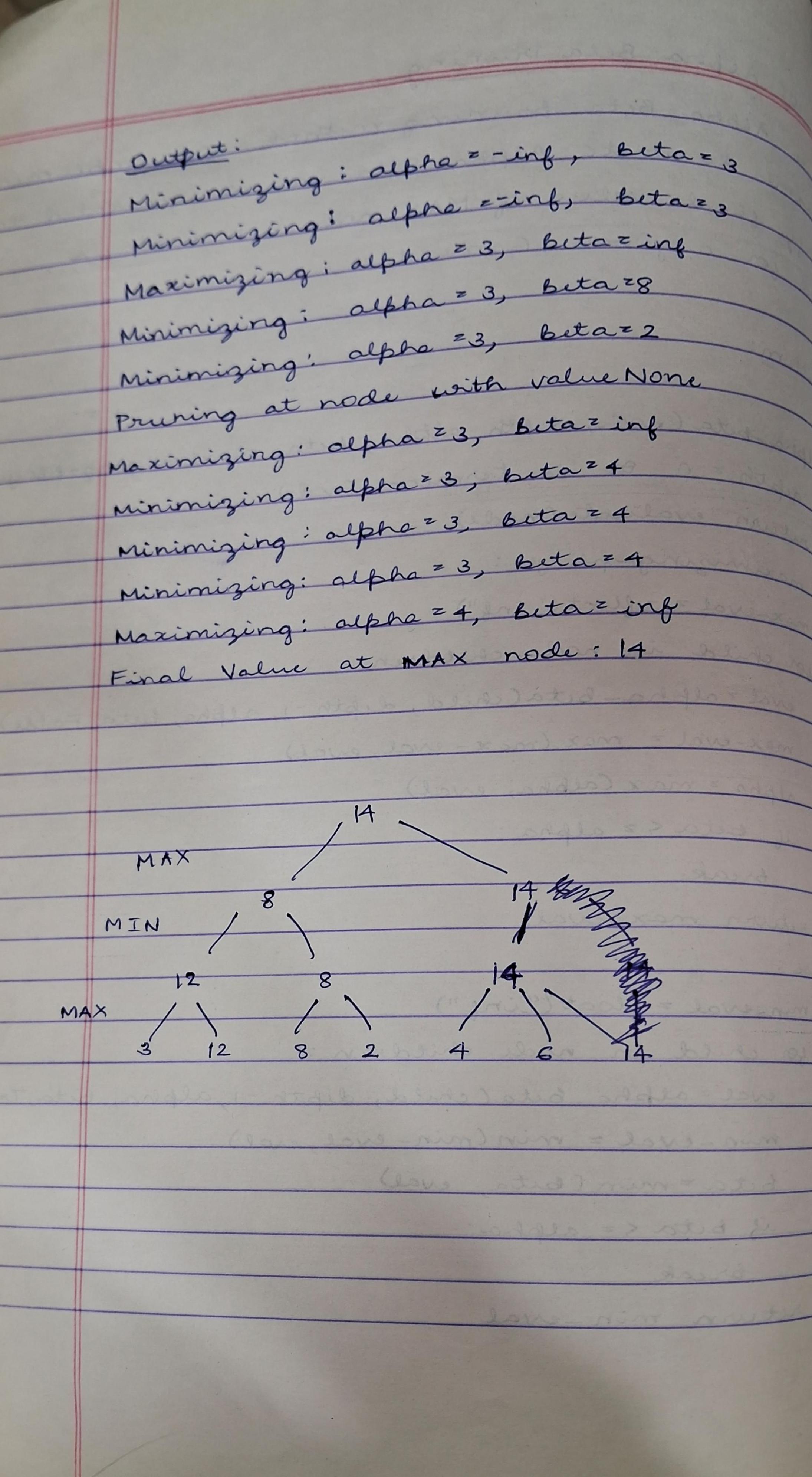
class Node:  
    def \_\_init\_\_(self, value=None, children=None):  
        self.value = value  
        self.children = children or []  
  
def alpha\_beta(node, depth, alpha, beta, maximizing\_player):  
    if depth == 0 or not node.children:  
        return node.value  
    if maximizing\_player:  
        max\_eval = -float('inf')  
        for child in node.children:  
            eval = alpha\_beta(child, depth - 1, alpha, beta, False)  
            max\_eval = max(max\_eval, eval)  
            alpha = max(alpha, eval)  
            print(f"Maximizing: alpha={alpha}, beta={beta}")  
            if beta <= alpha:  
                print(f"Pruning at node with value {node.value}")    
                break    
        return max\_eval  
    else:  
        min\_eval = float('inf')  
        for child in node.children:  
            eval = alpha\_beta(child, depth - 1, alpha, beta, True)  
            min\_eval = min(min\_eval, eval)  
            beta = min(beta, eval)  
            print(f"Minimizing: alpha={alpha}, beta={beta}")  
            if beta <= alpha:  
                print(f"Pruning at node with value {node.value}")    
                break    
        return min\_eval  
root = Node(children=[  
    Node(children=[Node(value=3), Node(value=12)]),  
    Node(children=[Node(value=8), Node(value=2)]),  
    Node(children=[Node(value=4), Node(value=6), Node(value=14)]),  
])  
final\_value = alpha\_beta(root, 3, -float('inf'), float('inf'), True)  
print(f"Final Value at MAX node: {final\_value}")

**Output:**

****

**Algorithm:**

****

****