Name: Nipun Chugh
UID: 22BCS10636
Class: KPIT - 901 / A

Q1. **Binary Tree Inorder Traversal**: Given the root of a binary tree, return the inorder traversal of its nodes' values.
Q2. **Binary Tree Preorder Traversal:** Given the root of a binary tree, return the preorder traversal of its nodes' values.
Q3. **Invert Binary Tree:** Given the root of a binary tree, invert the tree, and return its root.
Q4 Populating Next Right Pointers in Each Node.
Q5. **Binary Tree Zigzag Level Order Traversal**: Given the root of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between)

Solutions :

A1.

```
#include <iostream>
#include <vector>
#include <stack>
using namespace std;
struct TreeNode
{
   int val;
   TreeNode *left;
   TreeNode *right;
   TreeNode(int x) : val(x), left(NULL), right(NULL) {}
};
void inorderRecursive(TreeNode *root, vector<int> &result)
{
   if (root == NULL)
      return;
   inorderRecursive(root->left, result);
   result.push_back(root->val);
   inorderRecursive(root->right, result);
}
vector<int> inorderIterative(TreeNode *root)
{
   vector<int> result;
   stack<TreeNode *> stack;
   TreeNode *current = root;
   while (current != NULL || !stack.empty())
   {
      while (current != NULL)
      {
         stack.push(current);
         current = current->left;
      }
      current = stack.top();
      stack.pop();
      result.push_back(current->val);
      current = current->right;
   }
   return result;
}
void printVector(const vector<int> &vec)
{
```

```
    for (int val : vec)
    {
        cout << val << " ";
    }
    cout << endl;
}
int main()
{
    TreeNode *root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);
    vector<int> resultRecursive;
    inorderRecursive(root, resultRecursive);
    cout << "Recursive Inorder Traversal: ";
    printVector(resultRecursive);
    vector<int> resultIterative = inorderIterative(root);
    cout << "Iterative Inorder Traversal: ";
    printVector(resultIterative);
    return 0;
}
```

Output :

```
Recursive Inorder Traversal: 1 3 2
Iterative Inorder Traversal: 1 3 2


=== Code Execution Successful ===
```

A2.

```
#include <iostream>
#include <vector>
#include <stack>
struct TreeNode
{
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
                                        right(right) {}
};
class Solution
{
public:
    std::vector<int> preorderTraversal(TreeNode *root)
    {
        std::vector<int> result;
        if (!root)
            return result;
        std::stack<TreeNode *> stack;
        stack.push(root);
        while (!stack.empty())
        {
            TreeNode *node = stack.top();
```

```
            stack.pop();
            result.push_back(node->val);
            if (node->right)
                stack.push(node->right);
            if (node->left)
                stack.push(node->left);
        }
        return result;
    }
};
TreeNode *createTree()
{
    TreeNode *root = new TreeNode(1);
    root->right = new TreeNode(2);
    root->right->left = new TreeNode(3);
    return root;
}
int main()
{
    Solution solution;
    TreeNode *root = createTree();
    std::vector<int> result = solution.preorderTraversal(root);
    std::cout << "Preorder Traversal: ";
    for (int val : result)
    {
        std::cout << val << " ";
    }
    std::cout << std::endl;
    return 0;
}
```

Output :

```
Preorder Traversal: 1 2 3


=== Code Execution Successful ===
```

A3.

```
#include <iostream>
#include <vector>
#include <stack>
struct TreeNode
{
    int val;
    TreeNode *left;
    TreeNode *right;
    TreeNode() : val(0), left(nullptr), right(nullptr) {}
    TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
    TreeNode(int x, TreeNode *left, TreeNode *right) : val(x), left(left),
                                                       right(right) {}
};
class Solution
{
```

```
public:
    int sumOfNodes(TreeNode *root)
    {
        if (!root)
            return 0;
        return root->val + sumOfNodes(root->left) + sumOfNodes(root->right);
    }
    TreeNode *invertTree(TreeNode *root)
    {
        if (!root)
            return nullptr;
        TreeNode *temp = root->left;
        root->left = root->right;
        root->right = temp;
        invertTree(root->left);
        invertTree(root->right);
        return root;
    }
};
TreeNode *createTree()
{
    TreeNode *root = new TreeNode(4);
    root->left = new TreeNode(2);
    root->right = new TreeNode(7);
    root->left->left = new TreeNode(1);
    root->left->right = new TreeNode(3);
    root->right->left = new TreeNode(6);
    root->right->right = new TreeNode(9);
    return root;
}
void printTree(TreeNode *root)
{
    if (!root)
        return;
    printTree(root->left);
    std::cout << root->val << " ";
    printTree(root->right);
}
int main()
{
    Solution solution;
    TreeNode *root = createTree();
    std::cout << "Original tree (in-order): ";
    printTree(root);
    std::cout << std::endl;
    root = solution.invertTree(root);
    std::cout << "Inverted tree (in-order): ";
    printTree(root);
    std::cout << std::endl;
    return 0;
}
```

Output :

```
Original tree (in-order): 1 2 3 4 6 7 9
Inverted tree (in-order): 9 7 6 4 3 2 1


=== Code Execution Successful ===
```

A4.

```
#include <iostream>
using namespace std;
struct Node
{
   int val;
   Node *left;
   Node *right;
   Node *next;
   Node(int x) : val(x), left(NULL), right(NULL), next(NULL) {}
};
class Solution
{
public:
   void connect(Node *root)
   {
      if (root == NULL)
      {
         return;
      }
      Node *current = root;
      while (current != NULL)
      {
         Node *levelStart = current;
         Node *prev = NULL;
         while (levelStart != NULL)
         {
            if (levelStart->left)
            {
               if (prev)
               {
                  prev->next = levelStart->left;
               }
               prev = levelStart->left;
            }
            if (levelStart->right)
            {
               if (prev)
               {
                  prev->next = levelStart->right;
               }
               prev = levelStart->right;
            }
            levelStart = levelStart->next;
         }
```

```
            current = current->left;
        }
    }
};
void printLevels(Node *root)
{
    while (root)
    {
        Node *current = root;
        while (current)
        {
            cout << current->val << " ";
            current = current->next;
        }
        cout << "# ";
        root = root->left;
    }
    cout << endl;
}
int main()
{
    // Example: root = [1,2,3,4,5,null,7]
    Node *root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);
    root->right->right = new Node(7);
    Solution solution;
    solution.connect(root);
    printLevels(root);
    return 0;
}
}
```

Output :

```
1 # 2 3 # 4 5 7 #


=== Code Execution Successful ===
```

A5.

```
#include <iostream>
#include <queue>
#include <vector>
#include <deque>
using namespace std;
struct Node
{
    int val;
    Node *left;
    Node *right;
```

```
    Node(int x) : val(x), left(NULL), right(NULL) {}
};
class Solution
{
public:
    vector<vector<int>> zigzagLevelOrder(Node *root)
    {
        vector<vector<int>> result;
        if (root == NULL)
        {
            return result;
        }
        queue<Node *> q;
        q.push(root);
        bool leftToRight = true;
        while (!q.empty())
        {
            int levelSize = q.size();
            deque<int> currentLevel;
            for (int i = 0; i < levelSize; ++i)
            {
                Node *currentNode = q.front();
                q.pop();
                if (leftToRight)
                {
                    currentLevel.push_back(currentNode->val);
                }
                else
                {
                    currentLevel.push_front(currentNode->val);
                }
                if (currentNode->left)
                {
                    q.push(currentNode->left);
                }
                if (currentNode->right)
                {
                    q.push(currentNode->right);
                }
            }
            result.push_back(vector<int>(currentLevel.begin(),
                                         currentLevel.end()));
            leftToRight = !leftToRight;
        }
        return result;
    }
};
void printZigzagOrder(const vector<vector<int>> &zigzagOrder)
{
    for (const auto &level : zigzagOrder)
    {
        for (int val : level)
        {
            cout << val << " ";
        }
        cout << endl;
    }
}
```

```
int main()
{
    // Example: root = [3,9,20,null,null,15,7]
    Node *root = new Node(3);
    root->left = new Node(9);
    root->right = new Node(20);
    root->right->left = new Node(15);
    root->right->right = new Node(7);
    Solution solution;
    vector<vector<int>> zigzagOrder = solution.zigzagLevelOrder(root);
    printZigzagOrder(zigzagOrder);
    return 0;
}
}
```

Output :

```
3
20 9
15 7


=== Code Execution Successful ===
```