Binary Search Tree Operations (60 Marks)

You are required to implement a set of operations on a Binary Search Tree (BST). Each student will implement one of the operations listed below. The total marks for each operation are equal, and the exercise should be completed within 60 minutes.

Operation 1: Search for a Key in the BST (10 Marks)

Write a C function `search_bst(Node *root, int key)` that searches for a key in a binary search tree. The function should return `true` if the key is found and `false` otherwise.
In addition to this basic search, you should also:
- Track the path traversed during the search and print it as a list of node values.
- Return the number of nodes visited during the search.

Operation 2: Find the Height and Diameter of the Binary Tree (10 Marks)

Write a C function `find_height_and_diameter(Node *root, int *diameter)` to compute the height of the binary tree and the diameter of the tree.
- The height of a tree is the length of the longest path from the root to a leaf.
- The diameter of a tree is the length of the longest path between any two nodes in the tree, which may or may not pass through the root.

Operation 3: Iterative Pre-order Traversal with Subtree Sum (10 Marks)

Write a C function `pre_order_iterative_with_sum(Node *root, int *sum)` to perform an iterative pre-order traversal and calculate the sum of all node values during the traversal.
- The traversal should use a stack.
- While performing the traversal, calculate the sum of all node values visited.

Operation 4: Iterative In-order Traversal with Balanced Tree Check (10 Marks)

Write a C function `in_order_iterative_with_balance_check(Node *root)` to perform an iterative in-order traversal while checking whether the tree is balanced.
- A balanced tree is defined as one where the heights of the left and right subtrees of any node differ by at most 1.
- Use two stacks: one for the traversal and one for tracking balance checks.

Operation 5: Post-order Traversal with Node Deletion Counter (10 Marks)

Write a C function `post_order_iterative_with_deletion_count(Node *root, int *delete_count)` to perform an iterative post-order traversal while counting the number of leaf nodes deleted.

- Use two stacks to implement the iterative post-order traversal.
- Count how many leaf nodes are deleted during the traversal.

Operation 6: Delete a Node from the Binary Tree with Rotation (10 Marks)

Write a C function `delete_node_with_rotation(Node *root, int key)` to delete a node from the binary search tree while performing tree rotations when necessary.

- You need to handle the three cases of deletion:
 1. No children (leaf node).
 2. One child.
 3. Two children (use the in-order successor and possibly perform rotations).

General Instructions:

Each student must write a C program that implements the required operation.
Provide clear documentation for your functions, including comments explaining the key steps in your approach.
Ensure that your program is tested with various edge cases (e.g., empty tree, single-node tree, skewed tree, balanced tree, etc.).
Handle both iterative and recursive approaches where applicable.
Ensure correct memory management when performing deletions.
Submit your program with all necessary code files before the 60-minute mark.