## Exercise 01

Draw a diagram of a tree structure and label the following components:

- Root
- Branches
- Leaves
- Nodes (including internal nodes, leaf nodes, parent, and child nodes)
- Edges
- Levels
- Height
- Depth of nodes
- Node Degree

Explain the role of each component in the tree.

## Exercise 02

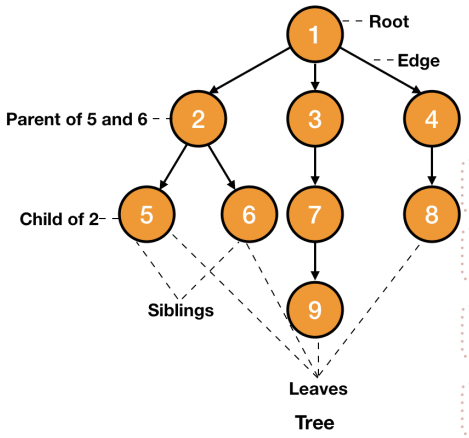
Implement a binary tree in C, including multiple methods for inserting nodes:

1. A function to create a new node.
2. Implement two different insertion methods:
    - Insertion based on given rules (e.g., smaller values to the left, larger to the right).
    - Insertion to the first available position (level-order insertion).
3. A function to visually display the tree in a readable format.

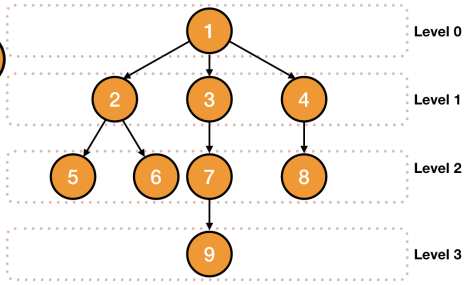
Test your implementation by inserting several nodes using both methods.

# Solution 01

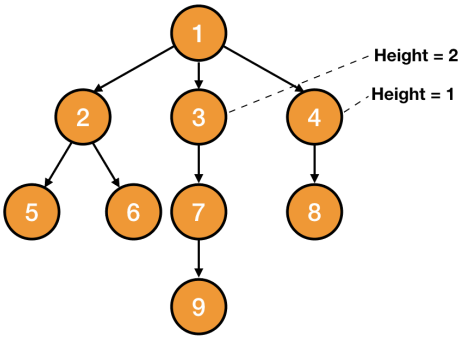
## Diagram and Descriptions



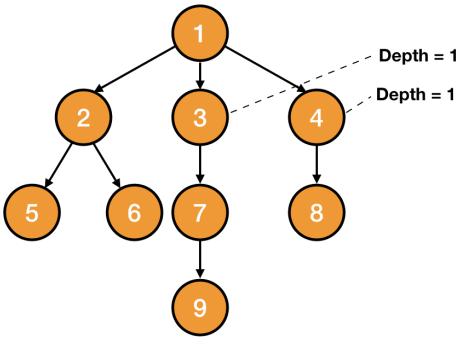

(a) Tree Structure 1



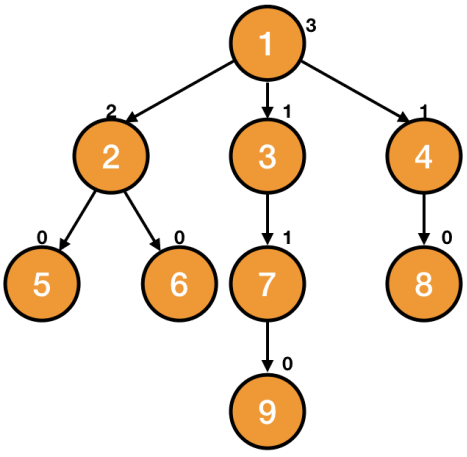

(b) Tree Structure 2




(c) Tree Structure 3




(d) Tree Structure 4



(e) Tree Structure 5

Figure 1: Multiple Tree Structures

- **Root**: The topmost node from where the tree originates.
- **Branches**: The connections between nodes, indicating parent-child relationships.
- **Leaves**: Nodes without children, indicating the endpoints of the tree.
- **Nodes**: Including:
  - **Internal Nodes**: Nodes with at least one child.
  - **Leaf Nodes**: Nodes without children.
  - **Parent Nodes**: Nodes that have children.
  - **Child Nodes**: Nodes that have a parent.
- **Edges**: Lines connecting nodes, representing pathways.
- **Levels**: The layers of the tree, with the root at level 0.
- **Height**: The length of the longest path from the root to a leaf.
- **Depth of Nodes**: The distance from the root to a node.
- **Node Degree**: The number of children a node has.

# Solution 02

## Binary Tree Implementation in C

```c
#include <stdio.h>
#include <stdlib.h>

// Define a queue structure
typedef struct Queue {
    int front, rear, size;
    unsigned capacity;
    struct Node** array;
} Queue;

// Define a Node structure
typedef struct Node {
    int data;
    struct Node* left;
    struct Node* right;
} Node;

// Function to create a new node
Node* createNode(int data) {
```

```
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->data = data;
    newNode->left = newNode->right = NULL;
    return newNode;
}
// Rule-based Insertion: Smaller values to the left, larger to the right
Node* insertByRule(Node* root, int data) {
    if (root == NULL) return createNode(data);
    if (data < root->data) root->left = insertByRule(root->left, data);
    else root->right = insertByRule(root->right, data);
    return root;
}
// Function to create a new queue
Queue* createQueue(unsigned capacity) {
    Queue* queue = (Queue*)malloc(sizeof(Queue));
    queue->capacity = capacity;
    queue->front = queue->size = 0;
    queue->rear = capacity - 1;
    queue->array = (Node**)malloc(capacity * sizeof(Node*));
    return queue;
}

// Function to check if the queue is full
int isFull(Queue* queue) {
    return (queue->size == queue->capacity);
}

// Function to check if the queue is empty
int isEmpty(Queue* queue) {
    return (queue->size == 0);
}

// Function to enqueue a node
void enqueue(Queue* queue, Node* item) {
    if (isFull(queue))
        return;
    queue->rear = (queue->rear + 1) % queue->capacity;
    queue->array[queue->rear] = item;
    queue->size = queue->size + 1;
}

// Function to dequeue a node
```

```c
Node* dequeue(Queue* queue) {
    if (isEmpty(queue))
        return NULL;
    Node* item = queue->array[queue->front];
    queue->front = (queue->front + 1) % queue->
    capacity;
    queue->size = queue->size - 1;
    return item;
}

// Level-order Insertion: Insert at the first
    available position
void insertLevelOrder(Node** root, int data) {
    Node* newNode = createNode(data);
    if (*root == NULL) {
        *root = newNode;
        return;
    }
    Queue* q = createQueue(100); // Create a queue for
    level-order traversal
    enqueue(q, *root);
    while (!isEmpty(q)) {
        Node* temp = dequeue(q);
        if (!temp->left) {
            temp->left = newNode;
            free(q);
            return;
        } else {
            enqueue(q, temp->left);
        }
        if (!temp->right) {
            temp->right = newNode;
            free(q);
            return;
        } else {
            enqueue(q, temp->right);
        }
    }
    free(q);
}

void printTree(Node* root, int space) {
    int i;
    if (root == NULL)
        return;
    space += 10;
```

```
    printTree(root->right, space);
    printf("\n");
    for (i = 10; i < space; i++)
        printf(" ");
    printf("%d\n", root->data);
    printTree(root->left, space);
}

int main() {
    Node* root = NULL;

    // Test Rule-based Insertion
    for (int i=0;i<=10;i++)
        root = insertByRule(root, i);

    printTree(root, 0);
    for (int i=0;i<=10;i++)
        printf("\n");

    // Reset tree and Test Level-order Insertion
    root = NULL;
    for (int i=0;i<=14;i++)
        insertLevelOrder(&root, i);

    printTree(root, 0);
    return 0;
}
```