Quiz 9.1	
Tree Structure	
Course Code: CPE010	**Program:** Computer Engineering
Course Title: Data Structures and Algorithms	**Date Performed:** 10/02/2025
Section: CPE21S4	**Date Submitted:** 10/02/2025
Name(s): Cabrera, Gabriel A.	**Instructor:** Engr. Jimlord Quejado

6. Output

General Instructions:

1. Define the following: (10)

A. Tree - A tree is a hierarchical data structure that consists of connected keys or nodes. It starts with one root node which can be connected to two or more child nodes below it, these children nodes can also be connected to one or more child nodes below them. Children nodes without any connection below them are called leaves.

B. Child node - A node is considered a child node when it is connected to a parent node, which is the node above the current node hierarchically.

C. Parent node - The parent node is a node which has one or more children connected below it.

D. Root node - The topmost node of a tree, this is where all the other nodes are eventually connected to.

E. Tree Traversal - Tree traversal is the process of visiting each node in a tree data structure exactly once in a specific order. Since trees are non-linear, there are multiple ways to visit the nodes, each serving a different purpose. The most common traversals are classified as depth-first which means going deep into a branch before moving to the next and breadth-first described as visiting all nodes at one level before moving to the next level.

F. Parse Tree - A parse tree, also known as a derivation tree, is an ordered, rooted tree that visually represents the syntactic structure of a string or program according to a formal grammar. Its nodes are labeled with strings or characters instead of numbers.

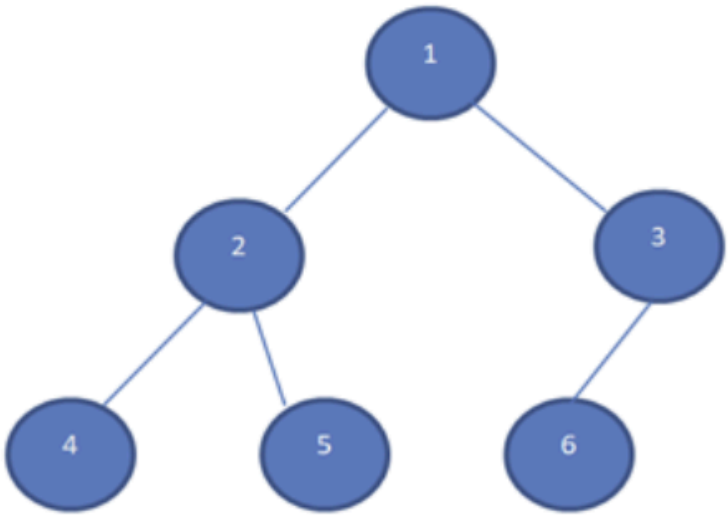
G. Order of Precedence - Order of precedence in trees refers to how operators are arranged in a tree structure based on their precedence and associativity. This is similar to math's PEMDAS operation which prioritizes the operations that came first in the order of operations.

H. Parsing - Parsing involves breaking down the input into smaller components (tokens) and organizing them into a structured format, such as a parse tree or syntax tree. This structured representation is essential for code analysis, generation, and overall improvement of the program.

I. Preorder Traversal - The pre-order traversal follows a Root -> Left -> Right order. It visits the current node first, then recursively traverses its left subtree, and finally its right subtree. This traversal is useful for creating a copy of a tree or for generating a prefix expression.

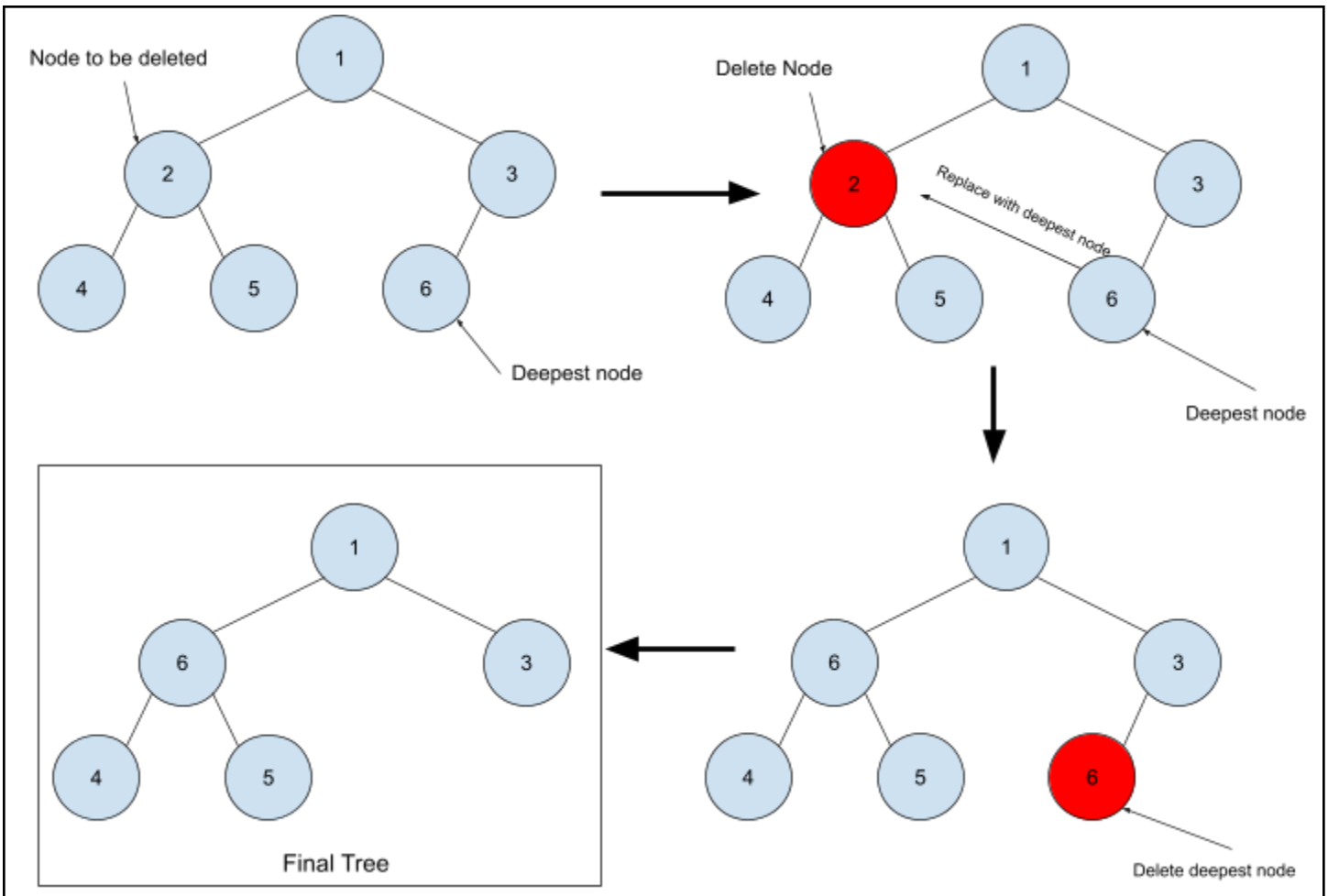
J. Inorder Traversal - This follows a Left->Root->Right order. The Inorder traversal is where the search starts from the bottom leftmost part of the tree, it then goes up and searches the right part of the left subtree, then goes up again and goes to the right subtree and searches the left part of the subtree and goes to the right again after. This loop continues until it searches the whole tree.

2. Identify the parts of the following tree. (10)



1. Root Node
2. Parent Node, left subtree
3. Parent Node, right subtree
4. Child node/leaf
5. Child node/leaf
6. Child node/leaf

3. Delete the node with an element value of 2 in the above tree structure. (10 pts)



4. Create a program in C++ to create a binary tree that looks like in the above figure.

main.cpp

Share Run

```cpp
#include <iostream>
struct Node {
    int data;
    Node* left;
    Node* right;
    Node (int val): data (val), left(nullptr), right(nullptr){}
};

class Binary{
private:
    Node* root;
    Node* insertNode(Node* node, int val){
        if (node==nullptr){
            return new Node(val);
        }
        if (val<node->data){
            node->left = insertNode(node->left,val);
        }
        else if (val>node->data){
            node->right = insertNode(node->right,val);
        }
        return node;
    }



    void BreadthFirst(Node* root) {
    if (root == nullptr) {
        return;
    }
```

```
32      std::cout << root->data << " ";
33      BreadthFirst(root->left);
34      BreadthFirst(root->right);
35      }
36  public:
37  Binary() : root(nullptr){}
38      void insert(int val){
39          root=insertNode(root,val);
40      }
41      void BreadthFirst(){
42          BreadthFirst(root);
43          std::cout<<std::endl;
44      }
45
46  };
47
48  int main(){
49      Binary tree;
50      tree.insert(1);
51      tree.insert(2);
52      tree.insert(3);
53      tree.insert(4);
54      tree.insert(5);
55      tree.insert(6);
56      std::cout<<"Binary Tree Breadth First Search\n";
57      tree.BreadthFirst();
58      return 0;
59  }
```

```
Binary Tree Breadth First Search
1 2 3 4 5 6
```

7. Assessment Rubric