CADT IDT

W8-S2 PRACTICE

Binary Search Tree

At the end of this practice, you should be able to…

- ✓ **Understand** and **Implement** Binary Search Tree (BST) Operations
- ✓ **Apply** Level-Order Traversal (BFS)
- ✓ **Analyze** Real-World Applications and Limitations of Binary Trees

• How to compile your code?

Assuming your file is named: exercise.cpp:

- ✓ Open a **terminal** at your file location
- ✓ **Compile** your Program using the following command

```
g++ -o exercise exercise.cpp
```

- ✓ **Run** Your Program using the following command

```
./exercise
```

How to submit?

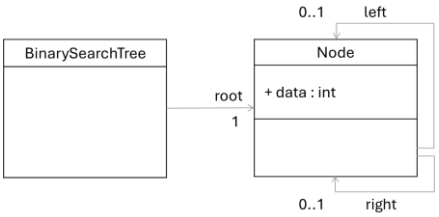
- ✓ **Push** your final code on **your GitHub repository**
- ✓ Then **attach the GitHub path** to the MS Team assignment and **turn it in**

No internet during this work

- ✓ No internet during this practice so you can **learn to solve problems on your own**.

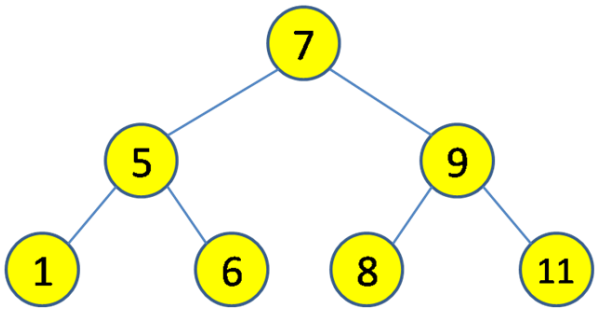
EXERCISE 1: Binary Search Tree (BST) Operations

- ✓ Implement the **Binary Search Tree (BST) ADT** from the UML
    1. BinarySearchTree contains a root node.
    2. Each Node stores an integer value and has references to its left and right children.
        - All nodes in the left subtree of a node have smaller values.
        - All nodes in the right subtree of a node have larger values.



UML – Binary Search Tree

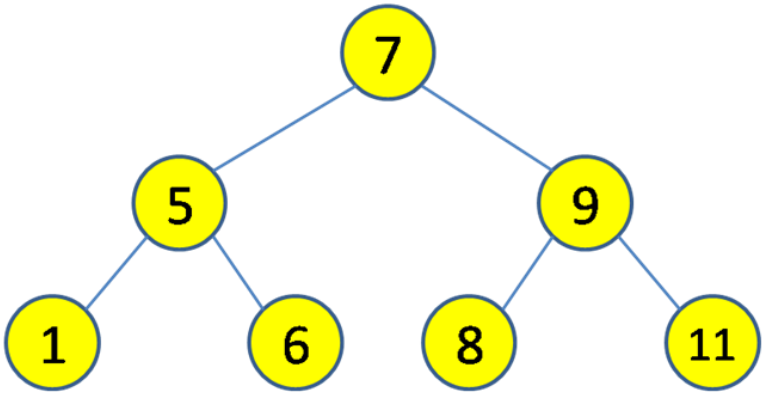
- ✓ Construct a binary tree to represent the tree below



- ✓ Implement methods for the following operations:
    1. Insert(): Insert a node in the correct position following the BST property.
    2. Search(): Search for a node with a specific value.
    3. Delete(): Delete a node from the BST and properly restructure the tree to maintain the BST property.
- ✓ *Using inOrder() to verify the structure of the BST*
- ✓ *Example or Test Case*
- ✓ *Note*
    - **main.cpp**: This is the primary file where you'll execute the test cases to validate the binary tree implementation. Additionally, the construction of the Binary Tree is also in this file.
    - ***BinarySearchTree.h***: This file contains the implementation of the Binary Search Tree class with the methods
    - ***Node.h***: This file contains the implementation of the Node class with the attribute.

EXERCISE 2: Level Order Traversal (Breadth First Search or BFS) of Binary Tree

- ✓ Objective
  1. Implement the **Level Order Traversal** (Breadth-First Search) for a binary tree.
  2. This traversal visits nodes level by level, from left to right.
- ✓ Step to complete
  1. Use your existing **BinaryTree** and **Node classes** to represent the tree structure.
  2. Construct a binary tree to represent the tree below:



3. Expected output:

Level-Order Traversal: 7 5 9 1 6 8 11

4. Level Order Traversal Implementation

```
levelOrder(root):
    create Queue q
    add root to q
    while q is not empty
        Node current = q.peek()
        q.dequeue()
        print current->data
        if current.left is not null:
            q.enqueue(current->left)
        if current.right is not null:
            q.enqueue(current->right)
```

EXERCISE 3: Identify Real-World Scenarios & Limitations

Objective

- ✓ In this exercise, you'll reflect on real-world scenarios involving **binary trees** or **N-ary** and analyze their use cases and limitations.
- ✓ The objective is to deepen your understanding of binary trees' advantages, limitations, and variations.

Instructions

- ✓ **Select a Scenario**: Choose a real-world scenario where **binary trees** or **N-ary trees** are commonly used. Some example scenarios include:
    - Database Indexing
    - File Systems
    - Decision Trees
    - And others (*you may propose your own*)
- ✓ **Identify the Use Case**
  For the scenario you selected:
    - ✓ Explain how **binary trees** or **N-ary trees** are used to solve the problem or address the task.
    - ✓ Discuss the advantages of using a tree in this context.
- ✓ **Analyze the Limitations**
    - ✓ Identify and discuss any potential limitations or challenges of using binary trees in that scenario.
    - ✓ Consider factors such as:
        - Performance issues
        - Access restrictions
        - Scalability concerns

Analysis Table

Complete the table below with the scenarios, use case and limitations you have identified.

Real-world Scenario	Use Case (How Binary Trees Solve the Problem)	Limitation (Challenges or Issues with Binary Trees)
Ex. Database Indexing	*Efficient search*	*Performance issues with unbalanced tree*
Contacts List	Organizing contacts alphabetically for faster searching.	Slower search if the list is not balanced.
Family Trees	Showing relationships between family members.	Hard to manage if the tree becomes very large or complex.
Game Menus	Structuring game options in a simple menu hierarchy.	Navigating deeply nested menus can take time.

Summary your findings here (if you want to further explain)