

# Assignment 4 on Binary Search Tree

## Overview

In this assignment, you will implement a basic Binary Search Tree (BST) using a linked list in C++. This assignment will help you understand how a basic binary search tree operates under the hood, and will guide you to implement the basic operations like insertion, deletion, search, etc., on a BST. You are provided with a project folder containing the following files:

- `BST.hpp` — The abstract base class with pure virtual functions defining the BST interface. **You must NOT modify this file.**
- `listBST.hpp` — The file where you will implement the BST functionality. You must write your code only inside the sections marked `// TODO:`.
- `task1.cpp` — Contains the `main()` function with predefined commands for testing the BST operations. You must implement this based on the input-output format as provided in the sample input-output for task 1.
- `in_task1.txt` — Sample input file containing necessary commands for BST operations for task 1.
- `out_task1.txt` — Corresponding expected output file for task 1.
- `task2.cpp` — Contains the `main()` function for simulating the Word Duel game as stated in task 2. You must implement this based on the input-output format as provided in the sample input-output for task 2.
- `in_task2.txt` — Sample input file for task 2.
- `out_task2.txt` — Corresponding expected output file for task 2.

Your BST implementation should work as expected when tested using the provided input and output files. Your implemented BST should be able to work for generic data types stored as key-value pairs, where the key and the value might differ in type. *You may safely assume that, if any user-defined data types (e.g. struct/class) are used, their comparator operators/functions will already be defined beforehand.* **You are NOT allowed to include any other libraries other than the ones already provided for you.**

## Task 1: Implementing the BST

You are required to complete the following:

1. Implement the `ListBST` class by filling in all `// TODO:` parts in `listBST.hpp`. You may define as many private helper functions as you need, but you must not add any extra `public` methods beyond what is defined in `BST.hpp`.
2. Complete the `main()` function in `task1.cpp` so that it parses the input file, processes the commands, and produces the output in the required format.
3. Refer to the function documentation inside `BST.hpp` and sample I/O files to ensure your implementations match the expected behavior.
4. Except printing the keys and values, there should not be any other print statement inside the `ListBST` class.

## Sample Input and Output

The table below shows a sample input (from `in_task1.txt`) and its expected output (from `out_task1.txt`). Your implementation should reproduce the same output for the given input. *If you notice any discrepancy between the following table and your provided I/O files, prioritize the provided files.*

Sample Input	Expected Output
F 1	Key 1 not found in BST.
E	Empty
I 8	Key 8 inserted into BST, BST (Default): (8:8)
I 10	Key 10 inserted into BST, BST (Default): (8:8 () (10:10))
I 3	Key 3 inserted into BST, BST (Default): (8:8 (3:3) (10:10))
M Min	Minimum value: 3
I 1	Key 1 inserted into BST, BST (Default): (8:8 (3:3 (1:1)) (10:10))
M Min	Minimum value: 1
M Max	Maximum value: 10
I 14	Key 14 inserted into BST, BST (Default): (8:8 (3:3 (1:1)) (10:10 () (14:14)))
M Max	Maximum value: 14
I 3	Insertion failed! Key 3 already exists in BST, BST (Default): (8:8 (3:3 (1:1)) (10:10 () (14:14)))
D 3	Key 3 removed from BST, BST (Default): (8:8 (1:1) (10:10 () (14:14)))
I 6	Key 6 inserted into BST, BST (Default): (8:8 (1:1 () (6:6)) (10:10 () (14:14)))
I 4	Key 4 inserted into BST, BST (Default): (8:8 (1:1 () (6:6 (4:4))) (10:10 () (14:14)))

I 13	Key 13 inserted into BST, BST (Default): (8:8 (1:1 () (6:6 (4:4))) (10:10 () (14:14 (13:13))))
I 7	Key 7 inserted into BST, BST (Default): (8:8 (1:1 () (6:6 (4:4) (7:7))) (10:10 () (14:14 (13:13))))
I 9	Key 9 inserted into BST, BST (Default): (8:8 (1:1 () (6:6 (4:4) (7:7))) (10:10 (9:9) (14:14 (13:13))))
T In	BST (In-order): (1:1) (4:4) (6:6) (7:7) (8:8) (9:9) (10:10) (13:13) (14:14)
T Pre	BST (Pre-order): (8:8) (1:1) (6:6) (4:4) (7:7) (10:10) (9:9) (14:14) (13:13)
T Post	BST (Post-order): (4:4) (7:7) (6:6) (1:1) (9:9) (13:13) (14:14) (10:10) (8:8)
D 8	Key 8 removed from BST, BST (Default): (9:9 (1:1 () (6:6 (4:4) (7:7))) (10:10 () (14:14 (13:13))))
T Pre	BST (Pre-order): (9:9) (1:1) (6:6) (4:4) (7:7) (10:10) (14:14) (13:13)
S	Size: 8
D 6	Key 6 removed from BST, BST (Default): (9:9 (1:1 () (7:7 (4:4))) (10:10 () (14:14 (13:13))))
S	Size: 7
D 10	Key 10 removed from BST, BST (Default): (9:9 (1:1 () (7:7 (4:4))) (14:14 (13:13)))
D 10	Removal failed! Key 10 not found in BST, BST (Default): (9:9 (1:1 () (7:7 (4:4))) (14:14 (13:13)))
F 4	Key 4 found in BST.
T In	BST (In-order): (1:1) (4:4) (7:7) (9:9) (13:13) (14:14)
T Pre	BST (Pre-order): (9:9) (1:1) (7:7) (4:4) (14:14) (13:13)
T Post	BST (Post-order): (4:4) (7:7) (1:1) (13:13) (14:14) (9:9)

## Task 2: Simulating Word Duel

Simulate a card game between two players, Phil and Claire, where each player aims to empty their hand of cards by discarding cards that match a randomly drawn word.

### Rules

#### 1. Setup:

- There exists an infinite set of cards, each with a unique word written on its back.
- At the start of the game, Phil and Claire are dealt  $n$  cards each, forming their initial hands. Each card in a player's hand is represented by a word, and multiple cards may contain the same word.
- The remaining cards form a draw pile.

#### 2. Gameplay:

- The game proceeds in turns.
- In each turn, one card is drawn at random from the draw pile, and its word is revealed.
- If Phil has one or more cards with the revealed word in his hand, he discards exactly one such card from his hand.
- Similarly, if Claire has one or more cards with the revealed word in her hand, she discards exactly one such card.
- Notice that if matched, both players discard their cards in the same turn.

### 3. Game End:

- The game ends when at least one player's hand becomes empty (i.e., they have no cards left).
- If both players' hands become empty in the same turn, the game is declared a tie.
- Otherwise, the player whose hand becomes empty first is declared the winner.

## Implementation Requirement

- You must use your implemented `ListBST` class to represent each player's hand, where the key is a word (string) and the value is the count of cards with that word (integer). Use of STL is NOT allowed.
- Modify the main function of the provided `task2.cpp` file.
- Follow the sample input (`in_task2.txt`) and output (`out_task2.txt`) for better understanding.
- The input starts with an integer  $n$ . The next two lines contain  $n$  words each, representing the initial hands of Phil and Claire respectively.
- Each of the following lines contains one word, representing the revealed word in each turn of the game. It is guaranteed that, the number of words provided in the input file will always be greater than or equal to the number of turns actually played in the game.
- The provided input file will guarantee a termination of the game.

## Compilation Instructions

- To compile and run your program for task 1, use:

```
g++ -std=c++11 task1.cpp -o task1
./task1 in_task1.txt > myout_task1.txt
```

- Compare your output (`myout_task1.txt`) with the provided `out_task1.txt`.

- To compile and run your program for task 2, use:

```
g++ -std=c++11 task2.cpp -o task2
./task2 in_task2.txt > myout_task2.txt
```

- Compare your output (myout\_task2.txt) with the provided out\_task2.txt.
- You may use Diffchecker for output comparison.

## Submission Guidelines

1. Create a directory named <your\_id> (e.g. 2305xxx).
2. Copy only the 3 modified files - listBST.hpp, task1.cpp and task2.cpp to the newly created directory.
3. Zip the directory and name it as <your\_id.zip> (e.g. 2305xxx.zip).
4. Do NOT submit BST.hpp and the provided sample I/O files.
5. Your code should be well-documented and follow proper indentation.

## Submission Deadline: Saturday, June 21, 2025, 10:00 PM

- Do not copy your code from your peers or other sources, which can result in **-100% penalty**.
- Do not use any AI tools (e.g. ChatGPT, Copilot etc.) in any part of the assignment.
- No submissions will be accepted after the deadline.

## Evaluation Policy

Task 1	Implementation of Insert, Remove, Find	$3 \times 5 = 15\%$
	Tree Traversal and Proper Printing	$4 \times 4 = 16\%$
	Implementation of Get, Update, FindMin, FindMax	$4 \times 4 = 16\%$
	Proper Deallocation of Memory	8%
	Implementation of Main and Proper Printing	15%
Task 2		30%

*Last updated on June 16, 2025.*