**Implementing a Level-Order BST Serialization System**

Develop a C++ program that uses a Binary Search Tree (BST) to store integer data. Implement level-order serialization and deserialization of the BST using file I/O and a queue data structure.

**Implementation Requirements**

1. **BST Implementation:**
   * Implement a template binary search tree that supports insertion, deletion, and search operations, and implementation of big rule of three
   * Use **smart pointers** in the BST implementation, not allowed to use raw pointers.
   * Implement two private methods **serializeLevelOrder()** and **serializeLevelOrder()** functions.
   * Implement two public methods **saveToFile()**to serialize the BST to a file, and **loadFromFile()** to rebuild the BST from a file.
2. **Queue Implementation:**
   * Implement a template queue data structure
   * Use the queue implementation to facilitate level-order traversal for serialization and deserialization.
3. **Level-Order Serialization:**
   * In **serializeLevelOrder(** , nodes are processed level by level. If a node is **null**, a placeholder (**"#"**) is added to the output to maintain the structure.
   * In **deserializeLevelOrder()**, the function reads values from the input string, reconstructing the tree by creating new nodes as necessary.
   * During serialization, represent null nodes with a specific character (e.g., **#**) and use commas as delimiters.
   * During deserialization, correctly interpret the null character placeholder and rebuild the tree structure accurately.
   * Example: The BST for the following serialized data is given below:

4,2,6,1,3,#,7,#,#,#,#,#,#

4

6

2

7

3

1

**Assignment Deliverables:**

1. **BST.h**: Header file for the template binary search tree implementation.
2. **Queue.h**: Header file for the template linked-based queue implementation
3. **main.cpp**: Main program file with user interaction for BST operations and demonstration of file I/O for serialization/deserialization. Sample run for main application is given below.

.

**Sample Run:**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **1**

Enter value to be inserted: **8**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **1**

Enter value to be inserted: **3**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **1**

Enter value to be inserted: **10**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **1**

Enter value to be inserted: **1**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **1**

Enter value to be inserted: **6**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **1**

Enter value to be inserted: **14**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **1**

Enter value to be inserted: **4**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **1**

Enter value to be inserted: **7**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **1**

Enter value to be inserted: **13**

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

**bst\_serialized.txt**

3. Deserialize BST from File

4. Search in the BST

8,3,10,1,6,#,14,#,#,4,7,13,#,#,#,#,#,#,#,

5. Exit

bst\_serialized.txt file created with the serialized data

Enter your choice: **2**

BST serialized and saved to file successfully.

Note that the file 'bst\_serialized.txt' will only be modified when the serialize option is actively chosen by the user.

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **4**

Enter a value:**10**

Search result: found

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **4**

Enter a value:**15**

Search result: not found

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **3**

Enter file name: **serialized.txt**

There is already **serialized.txt** file with the following content

4,2,6,1,3,#,7,#,#,#,#,#,#

BST deserialized from file successfully.

PreOrder: 4 2 1 3 6 7

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **4**

Enter a value: **2**

Search result: found

-- Binary Search Tree Operations --

1. Insert Element

2. Serialize BST to File

3. Deserialize BST from File

4. Search in the BST

5. Exit

Enter your choice: **5**

Exiting program.

**User Requirements:**

1. **BST Initialization:**
   * Instantiate a binary search tree object that holds integers.
2. **Menu Display and Input Handling:**
   * Display a menu with options to insert elements, serialize to file, deserialize from file, search the BST, and exit the program.
   * Prompt the user for a choice and handle the input to carry out the corresponding operation.
3. **Insert Element:**
   * Prompt the user to enter an integer value to insert into the BST.
   * Use the BST's **insert** method to add the value to the tree.
4. **Serialize BST to File:**
   * When this option is selected, serialize the BST and save it to a file with the default name "bst\_serialized.txt".
   * Ensure that serialization is done in a level-order (pre-order) and use appropriate node delimiters.
   * Handle any file I/O exceptions and inform the user of success or failure.
5. **Deserialize BST from File:**
   * Prompt the user for a file name to read the serialized BST data from.
   * Deserialize the data from the file and reconstruct the BST structure.
   * After successful deserialization, display the BST in pre-order to confirm the structure has been preserved.
   * Handle any file I/O or format exceptions and inform the user of success or failure.
6. **Search in the BST:**
   * Prompt the user to enter an integer value to search for in the BST.
   * Use the BST's **search** method to determine if the value is present in the tree.
   * Inform the user whether the search value was found or not.
7. **Exit:**
   * Allow the user to exit the program cleanly by selecting this option.

**Submission**

You are asked to submit your work as a single zip file via CANVAS. Zip file will include all source codes (.h and .cpp files) and also include Readme.txt where you explain what you’ve learned in this assignment.

Please use the following file format while naming the zip file:

LastNameFirstnameX\_Y.zip where LastNameFirstname is your last name with the first letter in capital, followed by your first name with the first letter in capital; the X is the course code; the Y is the assignment #. (ex: SerceFatmaCS300\_4.zip)

**Evaluation Criteria**

* BST Implementation [60 points]
* Queue Implementation [20 points]
* Main.cpp [20 points]

Criteria:

1. *Correctness of Implementation*: All functionalities should work without any errors or bugs.
2. *Code Organization and Structure*: The code should be well-organized, modular, and structured. Proper indentation and naming conventions should be followed.
3. *Comments and Documentation*: The code should be well-commented, explaining complex logic, tricky parts, and the purpose of functions/methods.