Lab 5A

Red Black Tree

In this lab session, we will continue to learn and implement an advanced tree data structure using a simple color-coding scheme to adjust the tree after each modification: **Red-Black Tree**.
Source code needs to be distributed in a single file named `StudentID_1.cpp`.

Each node in the Red-Black Tree has the following basic structure:

```
enum Color
{
    RED,
    BLACK
};

struct Node
{
    int key;
    Color color;
    Node* left;
    Node* right;
    Node* parent;
};
```

Implement these basic functions and operations on the Red-Black Tree as follows:

1. Create a new Node with a given value.

   `Node* newNode(int data)`

2. Insert a new value into a Red-Black Tree.

   `Node* insert(Node* root, int data)`

3. Search for a Node with a given value in the Red-Black Tree. Return `NULL` if not found.

   `Node* search(Node* root, int data)`

4. Delete a Node with a given value from the Red-Black Tree.

   `Node* deleteNode(Node* root, int data)`

5. Traversal in **Level-order**.

   `void LevelOrder(Node* root)`

Lab 5B

B Tree

In this lab session, we will learn and implement an advanced tree data structure: the **B-Tree**. Source code needs to be distributed in a single file named `StudentID_2.cpp`.

Each node in the B-Tree has the following basic structure:

```
const int MAX = 3; // B-Tree of order 3 (modifiable)

struct Node
{
    int keys[MAX];          // Array of keys
    Node* children[MAX+1]; // Pointers to child nodes
    int numKeys;            // Current number of keys
    bool isLeaf;            // Flag to denote leaf node
};
```

Implement the following basic functions and operations for the B-Tree:

1. Create a new node.

   `Node* newNode(bool isLeaf)`

2. Insert a new value into the B-Tree.

   `Node* insert(Node* root, int data)`

3. Search for a node containing a given value in the B-Tree. Return `NULL` if not found.

   `Node* search(Node* root, int data)`

4. Delete a node containing a given value from the B-Tree.

   `Node* deleteNode(Node* root, int data)`

5. Traversal in **Pre-order**, **In-order**, **Post-order** and **Level-order**.

   `void NLR(Node* root)`

   `void LNR(Node* root)`

   `void LRN(Node* root)`

   `void LevelOrder(Node* root)`

Submission

Your source code must be contributed in the form of a compressed file and named your submission according to the format `StudentID.zip`. Here is a detail of the directory organization:

```
StudentID
├── StudentID_1.cpp
└── StudentID_2.cpp
```

The end.