Activity: Implementing and Using a Binary Search Tree

Goal: The goal of this lab is to practice how to implement a generic Binary Search Tree (BST) using C++ templates.

Part I: Binary Search Tree Implementation

1. Create a file named **BST.h** and start defining a template for the BST as shown below.

```
template <class T>
struct BinaryNode {
   T data;
   BinaryNode<T>* left;
   BinaryNode<T> * right;
   BinaryNode<T>(T data): data(data), left(nullptr), right(nullptr) {}
};


class BST {
private:
    BinaryNode<T>* root;

public:
    BST():root(nullptr){}


};
```

2. Implement **insert()** method int the BST class to allow the insertion of elements:

Sample Call:

```
int main(){
   BST<int> bst;
   bst.insert(8);
   bst.insert(3);
   bst.insert(10);
   bst.insert(1);
   bst.insert(6);
   bst.insert(14);
   bst.insert(4);
   bst.insert(7);
   bst.insert(13);
   return 0;
}
```

3. Implement **inOrder()** in the BST to enable in-order traversal of the BST.

   Sample Call:

   ```
   int main(){
      BST<int> bst;
      bst.insert(8);
      bst.insert(3);
      bst.insert(10);
      bst.insert(1);
      bst.insert(6);
      bst.insert(14);
      bst.insert(4);
      bst.insert(7);
      bst.insert(13);

      return 0;
   }
   ```

   Sample Output:

   ```
   1 3 4 6 7 8 10 13 14
   ```

4. Implement **preOrder()** method in the BST so that the output of the following program is:

   Sample Call:

   ```
   //…
   int main(){
         BST<int> bst;
         bst.insert(8);
         bst.insert(3);
         bst.insert(10);
         bst.insert(1);
         bst.insert(6);
         bst.insert(14);
         bst.insert(4);
         bst.insert(7);
         bst.insert(13);
         bst.preOrder();
         return 0;
   }
   ```

   Sample Output:

   ```
   8 3 1 6 4 7 10 14 13
   ```

5. Implement **postOrder()** method in the BST so that the output of the following program is:

   Sample Call:

   ```
   //…
   int main(){
         BST<int> bst;
         bst.insert(8);
         bst.insert(3);
         bst.insert(10);
         bst.insert(1);
         bst.insert(6);
         bst.insert(14);
         bst.insert(4);
         bst.insert(7);
         bst.insert(13);
         bst.posOrder();
         return 0;
   }
   ```

   Sample Output:

   ```
         1 4 7 6 3 13 14 10 8
   ```

6. Implement **search()** method in the BST. The method returns true if the given item is in the BST.

   Sample Call:

   ```
   //…
   int main(){
      BST<int> bst;
      bst.insert(8);
      bst.insert(3);
      bst.insert(10);
      bst.insert(1);
      bst.insert(6);
      bst.insert(14);
      bst.insert(4);
      bst.insert(7);
      bst.insert(13);

      cout<<"Does 10 exist in the tree? "<<(bst.search(10) ? "Yes" : "No")<<endl;
      cout<<"Does 15 exist in the tree? "<<(bst.search(15) ? "Yes" : "No")<<endl;
      return 0;
   }
   ```

   Sample Output:

   ```
         Does 10 exist in the tree? Yes
         Does 15 exist in the tree? No
   ```

7. Implement **remove()** method in the BST so that the given item is removed from the BST

   Sample Call:

   ```
   //…
   int main(){
      BST<int> bst;
      bst.insert(8);
      bst.insert(3);
      bst.insert(10);
      bst.insert(1);
      bst.insert(6);
      bst.insert(14);
      bst.insert(4);
      bst.insert(7);
      bst.insert(13);
      bst.remove(10);
      bst.inOrder();
      return 0;
   }
   ```

   Sample Output:

   ```
         1 3 4 6 7 8 13 14
   ```

8. Implement **clear()** method in the BST so that nodes are deleted and the tree becomes empty.

   Sample Call:

   ```
   //…
   int main(){
      BST<int> bst;
      bst.insert(4);
      bst.insert(3);
      bst.insert(5);
      bst.insert(2);
      bst.insert(1);
      bst.clear();
      cout<<"Is the tree empty? "<<(bst.isEmpty() ? "Yes" : "No")<<endl;
      return 0;
   }
   ```

   Sample Output:

   ```
         Is the tree empty? Yes
   ```

Part II: Using Binary Search Tree to build library catalog by ISBN.

Develop a class **LibraryManager**, that reads book information from a file named **data.txt** and inserts each book into a binary search tree keyed by ISBN. After constructing the tree, perform an in-order traversal to display the book information sorted by ISBN.

Sample Call:

```
int main()
{
    LibraryManager library;
    library.loadFromFile("data.txt");
    library.displayBooks();
    cout<<library.searchBook("978-1-4493-0358-7");
    return 0;
}
```

Sample Output:

```
978-0-13-110362-7  The C Programming Language  Brian W. Kernighan
978-0-13-235088-4  Clean Code: A Handbook of Agile Software Craftsmanship  Robert C. Martin
978-0-201-63361-0  The C++ Programming Language  Bjarne Stroustrup
978-0-262-03308-4  Introduction to Algorithms  Thomas H. Cormen
978-0-321-57351-3  Design Patterns: Elements of Reusable Object-Oriented Software  Erich Gamma
978-0-596-52068-7  Programming Perl  Larry Wall
978-1-118-17305-7  Big Data: Principles and Best Practices  Nathan Marz
978-1-118-20691-1  Data Science from Scratch  Joel Grus
978-1-119-36644-7  Python for Data Analysis  Wes McKinney
978-1-4493-0358-7  Programming Python  Mark Lutz
978-1-4493-5573-0  You Don't Know JS: Scope & Closures  Kyle Simpson
978-1-4919-1889-0  Fluent Python  Luciano Ramalho
978-1-59327-584-6  Hacking: The Art of Exploitation  Jon Erickson
978-1-59327-950-9  Black Hat Python: Python Programming for Hackers and Pentesters  Justin Seitz
1
```