**DSA-FOIT** **Lab 10**

**Objectives:**

In this lab, students will practice:

1. Binary Search Trees
2. Recursive insert operation, recursive in-order pre-order post-order traversal, and some other recursive operations on BST
3. Iterative insert and Iterative in-order pre-order post-order traversal using stack

**Question 1**

Implement the following Tree Node:

```
template <typename k, typename v>
struct TNode
{
 k key;
 v value;
 TNode<k, v> *leftChild;
 TNode<k, v> *rightChild;
}
```

Now implement a binary search tree class "BST" which contains **root** of type TNode as data member. You have to implement the following member functions for your binary search tree:

a. A default Constructor which sets the root to nullptr.

b. A recursive "insertRec" function which is passed as parameter a key and a corresponding value. It then uses **recursion** to insert the <key, value> pair while considering the insertion rules. If the key already exists in the BST, it simply replaces the value.
```
void insertRec(k const  key, v const value)
```

c. A function "insert" which is passed as parameter a key and a corresponding value. It then **iteratively** inserts the <key, value> pair while considering the insertion rules. If the key already exists in the BST, simply replace the value.
```
void insert(k const  key, v const value)
```

d. A function "search" which is passed as parameter a key. The function then uses **recursion** to return pointer to the corresponding value. If the key does not exist, the function returns null.
```
v* search(k key)
```

e. A function "inorderPrintkeysRec" which prints the keys using **recursive** inorder traversal.
```
void inorderPrintKeysRec() const
```

f. A function "inorderPrintkeys" which prints the keys using **iterative** inorder traversal.
```
void inorderPrintKeys() const
```

g. A function "preorderPrintkeysRec" which prints the keys using **recursive** preorder traversal.
```
void preorderPrintKeysRec() const
```

h. A function “preorderPrintkeys” which prints the keys using **iterative** preorder traversal.
```
void preorderPrintKeys() const
```

i. A function “postorderPrintkeysRec” which prints the keys using **recursive** postorder traversal.
```
void postorderPrintKeysRec() const
```

j. A function “postorderPrintkeys” which prints the keys using **iterative** postorder traversal.
```
void postorderPrintKeys() const
```

k. A function “length” which uses **recursion** to return the count of total nodes in BST.
```
int length() const
```

l. A function “printAllAncestors” which is passed as parameter a key. The function then prints the keys of all ancestors of the node containing that key.
```
void printAllAncestors(k const  key) const
```

**Now run the following main program.**

```
int main()
{
	BST<int, int> tree; //the key and value both are of type int

	tree.insert(500, 500);
	tree.insertRec(1000, 1000);
	tree.insert(1, 1);
	tree.insert(600, 600);
	tree.insertRec(700, 700);
	tree.insert(10, 10);
	tree.insert(30, 30);
	tree.insertRec(9000, 9000);
	tree.insert(50000, 50000);
	tree.insertRec(20, 20);


	cout << "Printing keys using iterative in-order traversal: ";
	tree.inorderPrintKeys();

	cout << endl << endl << "Printing keys using recursive in-order traversal: ";
	tree.inorderPrintKeysRec();

	cout << endl << endl << "Printing keys using iterative pre-order traversal: ";
	tree.preorderPrintKeys();

	cout << endl << endl << "Printing keys using recursive pre-order traversal: ";
	tree.preorderPrintKeysRec();


	cout << endl << endl << "Printing keys using iterative post-order traversal: ";
	tree.postorderPrintKeys();
```

```
	cout << endl << endl << "Printing keys using recursive post-order traversal: ";
	tree.postorderPrintKeysRec();



	cout << endl << endl<< "Tree Length: " << tree.length() << endl << endl;

	int *val = tree.search(1);

	if (val != nullptr)
	{
		cout << "1 found" << endl;
	}

	val = tree.search(123);

	if (val == nullptr)
	{
		cout << "123 not found" << endl;
	}

	cout <<endl<< "Printing the keys of ancestor nodes of 20";
	tree.printAllAncestors(20);

	system("pause");
}
```