EE2410 Data Structure Coding HW #4 -- Trees (Chapter 5)
due date 5/26/2024, 23:59
You should submit:
(a) All your source codes (C++ file).
(b) Show the execution trace of your program, i.e., write a client main() to demonstrate all functions you designed using example data.

Submit your homework before the deadline (midnight of 5/26). Fail to comply (**late** homework) will have ZERO score. **Copy** homework will have ZERO score on both parties and SERIOUS consequences.

1. (40%) Binary tree

Develop a complete C++ template class for binary trees shown below.

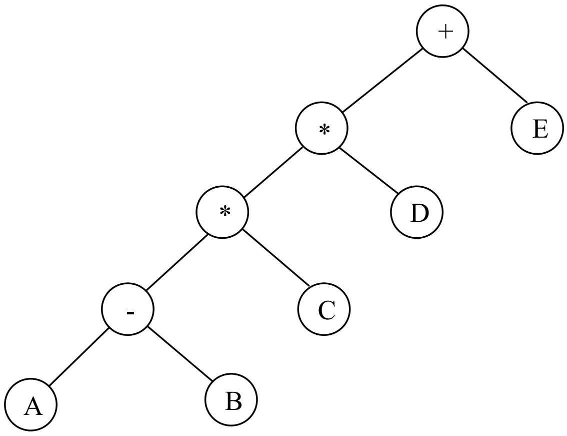
```
template <class T> class Tree;
template <class T>
class TreeNode {
friend class Tree <T>;
friend class InorderIterator<T>;//inorder iterator
private:
    T data;
    TreeNode <T> *leftChild;
    TreeNode <T> *rightChild;
};

template<class T>
class Tree
{
friend class InorderIterator<T>;//inorder iterator
public:
    Tree(); // constructor for an empty binary tree
    Tree(Tree<T>& bt1, T& item, Tree<T>& bt2);
    Tree(const Tree<T>&); //copy constructor
    // constructor given the root item and left subtrees bt1 and right subtree bt2
    ~Tree();
    bool IsEmpty(); // return true iff the binary tree is empty
    Tree<T> LeftSubtree(); // return the left subtree
    Tree<T> RightSubtree();// return the right subtree
```

```
        T RootData();    // return the data in the root node of *this
        // more operations
    private:
        TreeNode <T> *root;
        void Visit(TreeNode<T> *p){cout << p->data << "    ";}
    };
    template <class T>
    class InorderIterator{
    public:
        InorderIterator(){ currentNode = root;} // Constructor
        InorderIterator(Tree<T> tree):t(tree){ currentNode = t.root; }
        T* Next();
        T& operator *();
        bool    operator!=(const InorderIterator r)
    private:
        Tree<T> t;
        Stack<TreeNode<T>*> s;
        TreeNode<T> * currentNode;
    };
```

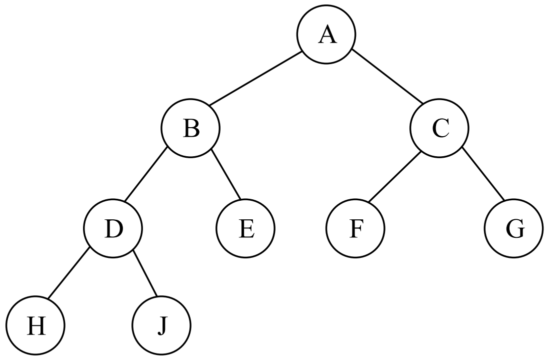
You must include a **constructor**, **copy constructor**, **destructor**, the traversal methods and operator overloads and the iterator class as shown below, and functions in **ADT 5.1**.

```
void Inorder()
void Preorder()
void Postorder()
void LevelOrder()
void NonrecInorder()
void NoStackInorder()
bool operator == (const Tree& t);
TreeNode<T> * Copy(TreeNode<T> * p); // Workhorse
bool Equal(const Tree<T>& t);
bool Equal(TreeNode<T>* a , TreeNode<T>* b);
void setup1();
void setup2();
void output();
```

Write 2 setup and display functions to establish and display 2 example binary trees shown below. Then **demonstrate** the functions you wrote.



(a)



(b)

sol:

Execution Trace:

```
sunpierce@pierces-MacBook-Air output % cd "/Users/sunpierce/C_C++/EE-DS/HW_4/output"
sunpierce@pierces-MacBook-Air output % ./"4.1_Binary_tree"
For the example binary tree (a):
└──+
    ├──*
    │   ├──*
    │   │   ├──-
    │   │   │   ├──A
    │   │   │   └──B
    │   │   └──C
    │   └──D
    └──E
The BT is nonempty.
The left subtree of (a):
└──*
    ├──*
    │   ├──-
    │   │   ├──A
    │   │   └──B
    │   └──C
    └──D
The right subtree of (a):
└──E
The root data is: +
Inorder traversal: A-B*C*D+E
Postorder traversal: AB-C*D*E+
Preorder traversal: +**-ABCDE
Levelorder traversal: +*E*D-CAB
Non-recursive inorder traversal: A-B*C*D+E
Inorder traversal without using stack: A-B*C*D+E
For the example binary tree (b):
└──A
    ├──B
    │   ├──D
    │   │   ├──H
    │   │   └──J
    │   └──E
    └──C
        ├──F
        └──G
The BT is nonempty.
The left subtree of (b):
└──B
    ├──D
    │   ├──H
    │   └──J
    └──E
The right subtree of (b):
└──C
    ├──F
    └──G
The root data is: A
Inorder traversal: HDJBEAFCG
Postorder traversal: HJDEBFGCA
Preorder traversal: ABDHJECFG
Levelorder traversal: ABCDEFGHJ
Non-recursive inorder traversal: HDJBEAFCG
Inorder traversal without using stack: HDJBEAFCG
BT(a) and BT(b) are not equal.
sunpierce@pierces-MacBook-Air output %
```

2. (20%) Threaded binary tree

Write a C++ template class for threaded binary trees: ThreadedTree according to the tree node structure as shown in Figure 5.21 in textbook. Then write C++ functions for:

(a) Forward iterator by sequencing through the nodes in inorder.

(b) Traverse a threaded binary tree in postorder.

(c) Traverse a threaded binary tree in preorder.

(d) Insert a new node r as the right child of node s in a threaded binary tree.

(e) Insert a new node l as the left child of node s in a threaded binary tree.

Use binary tree (b) shown above as example to construct a threaded binary tree and demonstrate the above five functions you implemented.

sol:

Execution Trace:

```
sunpierce@pierces-MacBook-Air output % cd "/Users/sunpierce/C_C++/EE-DS/HW_4/output"
sunpierce@pierces-MacBook-Air output % ./"4.2_ThreadedBT"
 (a) Forward iterator by sequencing through the nodes in inorder: HDJBEAFCG
 (b) Traverse a threaded binary tree in postorder: HJDEBFGCA
 (c) Traverse a threaded binary tree in preorder: ABDHJECFG
 (d) Suppose we want to insert a new node r as the right child of node s in a threaded binary tree.
 Enter the data of the node s: B
 Enter the data of the node r: Q
 The inorder traversal of the new tree: HDJBQEAFCG
 (e) Suppose we want to insert a new node l as the left child of node s in a threaded binary tree.
 Enter the data of the node s: Q
 Enter the data of the node l: X
 The inorder traversal of the new tree: HDJBXQEAFCG
sunpierce@pierces-MacBook-Air output %
```

3. (20%)

(a) Write a C++ class MaxHeap that derives from the abstract base class in **ADT 5.2 MaxPQ** and implement all the virtual functions of MaxPQ.

ADT 5.2 MaxPQ
template <**class** T>
class MaxPQ {
public:
    virtual ~MaxPQ() {} // virtual destructor
    virtual bool IsEmpty() **const** = 0; //return **true** iff empty
    virtual const T& Top() **const** = 0; //return reference to the max
    virtual void Push(**const** T&) = 0;
    virtual void Pop() = 0;
};

The class MaxHeap should include a **bottom up heap construction initialization** function, the push function for inserting a new key and pop function for deleting and the max key. You should also write a client function (main()) to demonstrate how to construct a max heap from a sequence of 13 integer number: 50, 5, 30, 40, 80, 35, 2, 20, 15, 60, 70, 8, 10 by using a series of 13 pushes and by bottom up initialization. Add necessary code for displaying your result.

(b) Write a C++ abstract class similar to ADT 5.2 for the ADT **MinPQ**, which defines a min priority queue. Then write a C++ class MinHeap that derives from this abstract class and implement all the virtual functions of MinPQ.

The class MinHeap should include a **bottom up heap construction initialization** function, the push function for inserting a new key and pop function for deleting and the min key. You should also write a client function (main()) to demonstrate how to construct a min heap from a sequence of 13 integer number: 50, 5, 30, 40, 80, 35, 2, 20, 15, 60, 70, 8, 10 by using a series of 13 pushes and by bottom up initialization. Add necessary code for displaying your result.

sol:

Execution Trace:

(a)

```
sunpierce@pierces-MacBook-Air output % cd "/Users/sunpierce/C_C++/EE-DS/HW_4/output"
sunpierce@pierces-MacBook-Air output % ./"4.3.a_MaxHeap"
Initializing max heap by using a series of 13 pushes: 80  70  35  20  60  30  2  5  15  40  50  8  10
How many element do you want to pop from the max heap? 5
Max heap after pop: 35  20  30  15  10  8  2  5
Initializing max heap by bottom up heap construction: 80  70  35  40  60  30  2  20  15  50  5  8  10
Max heap is non-empty
Enter a number to push to the max heap: 18
Max heap after inserting: 80  70  35  40  60  30  18  20  15  50  5  8  10  2
The max element of is: 80
sunpierce@pierces-MacBook-Air output %
```

(b)

```
sunpierce@pierces-MacBook-Air output % cd "/Users/sunpierce/C_C++/EE-DS/HW_4/output"
sunpierce@pierces-MacBook-Air output % ./"4.3.b_MinHeap"
Initializing min heap by using a series of 13 pushes: 2  15  5  20  60  8  30  50  40  80  70  35  10
How many element do you want to pop from the max heap? 5
Min heap after pop: 20  40  30  50  60  35  70  80
Initializing min heap by bottom up heap construction: 2  5  8  15  60  10  30  20  40  80  70  35  50
Max heap is non-empty
Enter a number to push to the max heap: 4
Max heap after inserting: 2  5  4  15  60  10  8  20  40  80  70  35  50  30
The min element of is: 2
The elements of min heap in ascending order: 2  4  5  8  10  15  20  30  35  40  50  60  70  80
sunpierce@pierces-MacBook-Air output %
```

4. (20%)

A Dictionary abstract class is shown in **ADT5.3 Dictionary**. Write a C++ class BST that derives from Dictionary and implement all the virtual functions. In addition, also implement

Pair<K, E>* RankGet(**int** r),

void Split(**const** K& k, BST<K, E>& small, pair<K, E>*& mid, BST<K, E>& big)

ADT5.3 Dictionary

template <**class** K, **class** E>

class Dictionary {

public:

  virtual bool IsEmptay() **const** = 0; // return true if dictionary is empty

  virtual pair <K, E>* Get(const K&) **const** = 0;

  // return pointer to the pair w. specified key

  virtual void Insert(**const** Pair <K, E>&) = 0;

  // insert the given pair; if key ia a duplicate, update associate element

  virtual void Delete(**const** K&) = 0; // delete pair w. specified key

};

Use a sequence of 13 integer number: 50, 5, 30, 40, 80, 35, 2, 20, 15, 60, 70, 8, 10 as 13 key values (type int) to generate 13 (key, element) (e.g., element can be simple char) pairs to construct the BST. Demonstrate your functions using this set of records.

sol:

Execution Trace:

```
sunpierce@pierces-MacBook-Air output % cd "/Users/sunpierce/C_C++/EE-DS/HW_4/output"
sunpierce@pierces-MacBook-Air output % ./"4.4_BST"
Level order traversal of the BST: (50,A) (5,B) (80,E) (2,G) (30,C) (60,J) (20,H) (40,D) (70,K) (15,I) (35,F) (8,L) (10,M)
BST is non-empty.
Enter a rank value: 5
The corresponding element is I
Enter a key value: 35
The corresponding element is F
Enter a key value to split: 30
Level order traversal of small BST: (5,B) (2,G) (20,H) (15,I) (8,L) (10,M)
Mid pair: (30,C)
Level order traversal of big BST: (50,A) (40,D) (80,E) (35,F) (60,J) (70,K)
Enter a key value to delete in the big BST: 80
Big BST after delete: (50,A) (40,D) (60,J) (35,F) (70,K)
sunpierce@pierces-MacBook-Air output %
```