EECS2040 Data Structure Hw #4 (Chapter 5 Tree)
due date 5/22/2022, 23:59

Format: Use a text editor to type your answers to the homework problem. You need to submit your HW in an HTML file or a DOCX file named as **Hw4-SNo.docx** or **Hw4-SNo.html**, where SNo is your student number. Submit the **Hw4-SNo.docx or Hw4-SNo.html** file via eLearn. Inside the file, you need to put the **header and your student number, name (e.g., EECS2040 Data Structure Hw #4 (Chapter 5 of textbook) due date 5/22/2022 by SNo, name)** first, and then the **problem** itself followed by your **answer** to that problem, one by one. The grading will be based on the correctness of your answers to the problems, and the **format**. Fail to comply with the aforementioned format (file name, header, problem, answer, problem, answer,…), will certainly degrade your score. If you have any questions, please feel free to ask me.

Part 2 Coding (5% of final Grade)

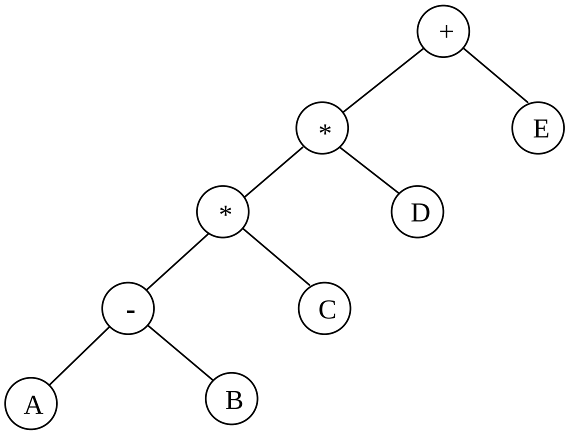
You should submit:
(a) All your source codes (C++ file).
(b) Show the execution trace of your program, i.e., write a client main() to demonstrate all functions you designed using example data..

1. (30%) Develop a complete C++ template class for binary trees shown in **ADT 5.1**. You must include a **constructor**, **copy constructor**, **destructor**, the traversal methods as shown below, and functions in **ADT 5.1**.
   void Inorder()
   void Preorder()
   void Postorder()
   void LevelOrder()
   void NonrecInorder()
   void NoStackInorder()
   bool operator == (**const** BinaryTree& t) **const**

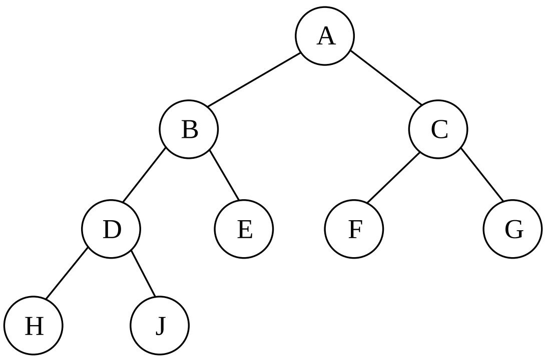
   ADT 5.1 BinaryTree
   template<**class** T>
   class BinaryTree
   { // objects: A finite set of nodes either empty or consisting
     // of a root node, left BinaryTree and right BinaryTree
   public:
       BinaryTree(); // constructor for an empty binary tree
       bool IsEmpty(); // return true iff the binary tree is empty

```
        BinaryTree(BinaryTree<T>& bt1, T& item, BinaryTree<T>& bt2);
        // constructor given the root item and left subtrees bt1 and right subtree bt2
        BinaryTree<T> LeftSubtree(); // return the left subtree
        BinaryTree<T> RightSubtree();// return the right subtree
        T RootData();    // return the data in the root node of *this
};
```

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



(a)



(b)

2. (35%) (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.

3. (35%) 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.