EECS2040 Data Structure Hw #4 (Chapter 5 Tree)
due date 5/30/2021, 23:59
by 107061123 孫元駿

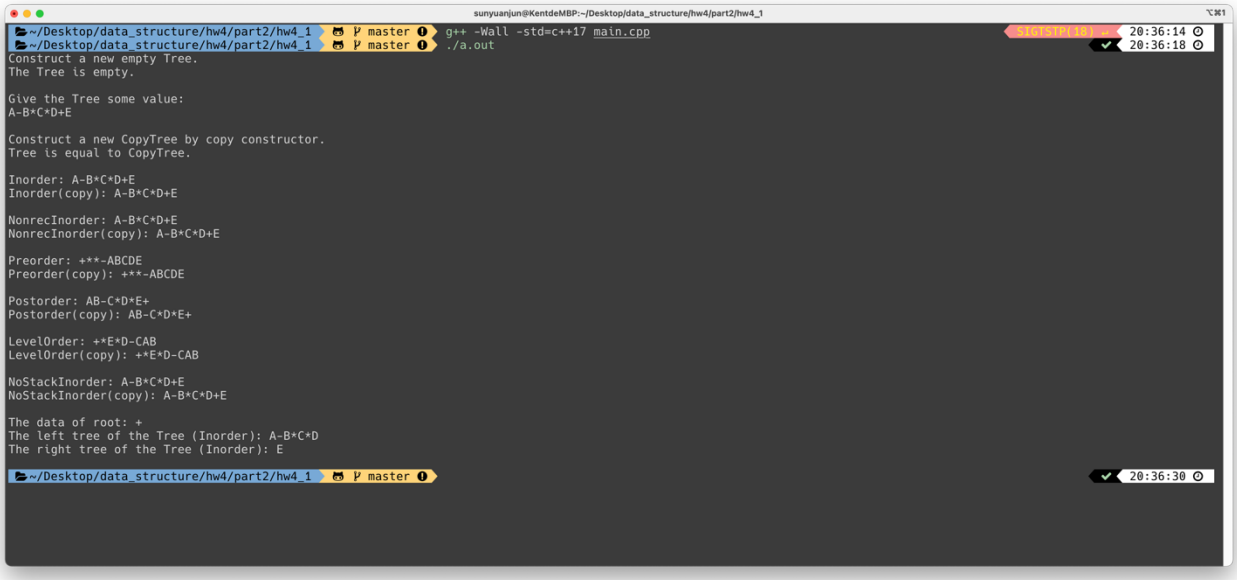
Part 2 Coding

You should submit:
(a) All your source codes (C++ file).
(b) Show the execution trace of your program.

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 four traversal methods, functions in **ADT 5.1**, … as shown below.
   void Inorder()
   void Preorder()
   void Postorder()
   void LevelOrder()
   void NonrecInorder()
   void NoStackInorder()
   bool operator == (**const** BinaryTree& t) **const**

   Write 2 setup functions to establish 2 example binary trees (use the two trees in Part 1 Question 5). Then **demonstrate** the functions you wrote.

```
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
};
```



The execution trace of my program:

First construct a empty tree T and show if the tree is empty by T.IsEmpty().

Then give T some value. And create a new CopyTree CT to copy T by the copy constructor CT = Tree(T) and check if T and CT are equal by T == CT.

Then, show Inorder, NonrecInorder, Preorder, Postorder, LevelOrder and NonStackInorder of T and CT.

Last, show the root of T by T.RootData(), and left child and right child by T.LeftChild() and T.RightChild() and show both tree (Inorder).

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;
};
```

(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.

Use Part1 Q6 example to demonstrate your program.

The execution trace of my program:

First, construct a max heap Q and a min heap H, and show if Q and H are all empty tree by

Q.IsEmpty() and H.IsEmpty().
Then, insert some value into Q and H using Part1 Q6 example by Q.Push(<element>) and H.Push(<element>). And show Q and H by Q.show() and H.show().
Then, pop one element from Q and pop one from H, and show the max element of Q by Q.Top() and show the min element of H by H.Top().
Last, show Q and P (after pop out one element).

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 Part1 Q6 example as 11 key values (type int) to generate 11 (key, element) (e.g., element can be simple char) pairs to construct the BST. Demonstrate your functions using this set of records.

The execution trace of my program:
First, construct a empty BST call T and check if it is empty by T.IsEmpty().

Then, insert some elements(key, value) to T by T.Insert(<key>, <value>) and print the tree by postorder type by T.Postorder(). Then get the node of key is 70 by T.Get(70) and get the node that it is rank 3 by T.RankGet(3).
Then, delete the elements that keys are 50, 5 and 60 by T.Delete(50), T.Delete(5) and T.Delete(60). After that show the postorder and the node that its key is 70 and the node that it is rank 3 like above.
Last, split the tree at key equal to 35 by T.Split(35, Small, Mid, Big), and store to different BST tree, and show the mid element, and the Small BST and Big BST.