CSCI 2720
Assignment 2 – Binary Search Trees
DUE: October 29th, 2018 11:59PM

Summary:

In this assignment you will create a Binary Search Tree to store and retrieve objects of type ItemType. The purpose of this assignment is for you to become familiar with basic tree operations, and understand the efficiency of trees compared to previously studied data structures. Binary Tree nodes have only two children, left and right. Nodes are compared based on their Key instance variable, which in this assignment is of type ItemType. All elements in the left subtree of a given node have key values less than that nodes', and likewise, all elements in the right subtree have key values greater. Duplicates are not allowed in this assignment. A Binary tree must maintain this sorted property at all times. In order to implement this BinaryTree, you will use the ItemType class created in earlier assignments, and note that ItemType does not require any additional functions. You may choose to implement any of the functions in Binary Tree class either iteratively or recursively. As with any assignment, you may create additional functions that helps you to implement the core functionality.(Hint: You will have to create additional functions if you choose the recursive approach outlined in the book.) Once all functions have been implemented for the Binary Tree class, you will create a Main application that initializes a tree based on file input, and allows the user to interactively modify it based on given commands. Finally, be sure to properly document all code with comments, and ensure that your output matches the example output exactly.

Project Files:

- ItemType.h and ItemType.cpp:
  Implementation remains the same as previous assignments.

- BinaryTree.h:
  Instance Variables:
    Node *root;
    int length;
  Public member functions:
    BinaryTree();
      Pre-Condition: None.
      Post-Condition: Tree is initialized.

    ~BinaryTree();
      Pre-Condition: Tree has been initialized.
      Post-Condition: All node pointers freed, root points to null, and length equals 0.

    void insert(ItemType &key);
      Pre-Condition: Tree and parameter key initialized.

Post-Condition: Insert a node with the value of key into the tree. No duplicates allowed.

void deleteItem(ItemType &key);
Pre-Condition: Tree and parameter key initialized.
Post-Condition: Remove a node with a key value equal to the parameter key's value and decrement length, otherwise leave tree unchanged. In situations, where the node to be deleted has two children, replace the deleted node with it's immediate predecessor.

void retrieve(ItemType &item bool &found) const;
Pre-Condition: Tree, key, and found are all initialized.
Post-Condition: Parameter key is set to equal the key of a node found in the tree, where the value instance variables of both keys are equal. Found is set to true if a node in the tree key value equal the parameter key's value, and false otherwise.

void preOrder() const;
Pre-Condition: The tree has been initialized.
Post-Condition: Print out the tree in pre-order.

void inOrder() const;
Pre-Condition: The tree has been initialized.
Post-Condition: Print out the the tree in in-order.

void postOrder() const;
Pre-Condition: The tree has been initialized.
Post-Condition: Print out the tree in post-order.

int getLength() const;
Pre-Condition: The tree has been initialized.
Post-Condition: Return value of length.

- BinaryTree.cpp:
  Implement all the functions defined in the header file.
- Main.cpp:
  Create a main application that matches example output exactly.

Example Output:

```
./main input.txt
Commands - insert (i), delete (d), retrieve (r), length (l), in-order (n), pre-order (p), post-order (o), quit
(q)
Enter a command: p
Pre-Order: 25 15 10 4 12 22 18 24 50 35
Enter a command: n
In-Order: 4 10 12 15 18 22 24 25 35 50
Enter a command: o
Post-Order: 4 12 10 18 24 22 15 35 50 25
Enter a command: l
List Length: 10
Enter a command: q
Quitting program…

./main input.txt
Commands - insert (i), delete (d), retrieve (r), length (l), in-order (n), pre-order (p), post-order (o), quit
(q)
Enter a command: i
Item to insert: 13
In-Order: 4 10 12 13 15 18 22 24 25 35 50
Enter a command: p
Pre-Order: 25 15 10 4 12 13 22 18 24 50 35
Enter a command: i
Item to insert: 25
Item alredy in tree.
In-Order: 4 10 12 13 15 18 22 24 25 35 50
Enter a command: q
Quitting program…

./main input.txt
Commands - insert (i), delete (d), retrieve (r), length (l), in-order (n), pre-order (p), post-order (o), quit
(q)
Enter a command: d
Item to delete: 50
In-Order: 4 10 12 15 18 22 24 25 35
Enter a command: o
Post-Order: 4 12 10 18 24 22 15 35 25
Enter a command: p
Pre-Order: 25 15 10 4 12 22 18 24 35
Enter a command: d
Item to delete: 100
Item not in tree.
In-Order: 4 10 12 15 18 22 24 25 35
Enter a command: q
Quitting program…
./main input.txt
```

```
Commands - insert (i), delete (d), retrieve (r), length (l), in-order (n), pre-order (p), post-order (o), quit
(q)
Enter a command: r
Item to be retrieved: 25
Item found in tree.
Enter a command: r
Item to be retrieved: 101
Item not in tree.
Enter a command: q
Quitting program…
```

```
./main input.txt
Commands - insert (i), delete (d), retrieve (r), length (l), in-order (n), pre-order (p), post-order (o), quit
(q)
Enter a command: k
Command not recognized. Try again
Enter a command: b
Command not recognized. Try again
Enter a command: q
Quitting program...
```

Grading Rubric:

Implementation	Grade
Binary Tree	85%
Insert	15%
Delete	25%
Pre-Order	10%
In-Order	10%
Post-Order	10%
Retrieve	5%
Get Length	5%
BinaryTree Deconstructor	5%
Main Application:	10%
Documentation & Comments	5%
Total:	100%

Compiling:

A Makefile should compile your code into program called “main”.

Your program should be run with the following command syntax:

```
./main <input file name>
```

Note: Code that fails to compile will receive a grade of a zero.

Late Policy:

Except in the cases of serious illness or emergencies, projects must be submitted before the specified deadline in order to receive full credit. Projects submitted late will be subject to the following penalties:

- If submitted 0–24 hours after the deadline 20% will be deducted from the project score
- If submitted 24–48 hours after the deadline 40% points will be deducted from the project score.
- If submitted more than 48 hours after the deadline a score of 0 will be given for the project.

Note that if your assignment is even one second late, it is considered late, and corresponding penalties will be applied. Students unable to submit a project due to a serious illness or other emergency should contact the instructor as soon as possible before a project’s deadline. Based upon the circumstances, the instructor will decide an appropriate course of action.

Submission:

Create a Readme file with the instructions on how to compile your program. Submit the following files within a zipped folder on eLC:

ItemType.h
ItemType.cpp
BinaryTree.h
BinaryTree.cpp
Main.cpp
Makefile
Readme