Programming Assignment 6: Morse Code Lookup BST

Assigned: Wednesday, March 8, 2017
Due: Wednesday, March 22, 2017 by midnight

I. Learner Objectives:

At the conclusion of this programming assignment, participants should be able to:

- Design, implement, and test a Binary Search Tree (BST)
- Apply a BST for looking up Morse Codes
- Discuss *classes* versus *objects*
- Implement *container* classes

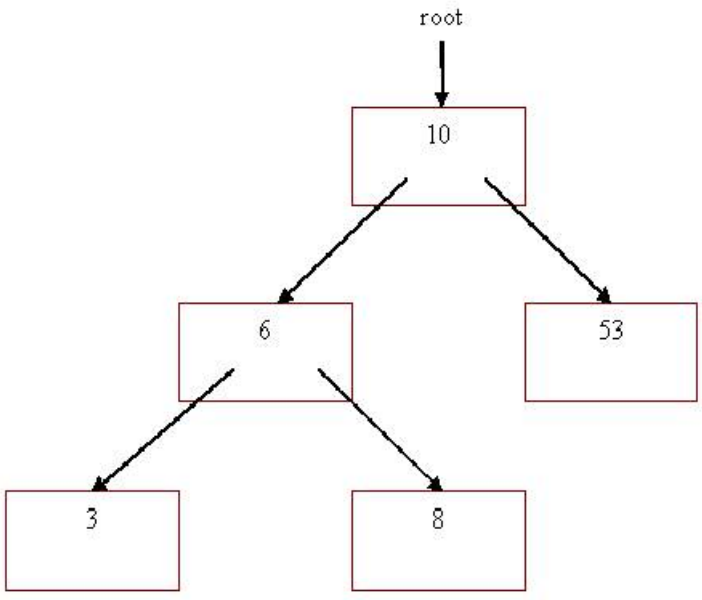
II. Prerequisites:

Before starting this programming assignment, participants should be able to:

- Analyze a basic set of requirements for a problem
- Compose basic C++ language programs
- Create basic test cases for a program
- Apply arrays, strings, and pointers
- Design, implement, and apply classes
- Design, implement, and apply linked lists

III. Overview & Requirements:

Recall, a Binary Search Tree (BST) data structure is a nonlinear data structure. A BST is traversed by starting at the root pointer. The root node is the first node inserted into the tree. Nodes are inserted into the tree such that all items to the left of the root node are less than, and all items to the right of the root are greater than its item. Also, this property holds true for any particular node in the tree. We will visualize a BST in the following way:



In this assignment you will be using a BST to convert English characters to Morse code strings. Morse code is a famous coding scheme that assigns a series of dots and dashes to each letter of the alphabet, each digit, and a few special characters. In

sound-oriented systems, the dot represents a short sound and the dash represents a long sound. Other representations of dots and dashes are used with light-oriented systems and signal-flag systems (from Deitel and Deitel C How to Program).

1. (15 pts) Defining the BSTNode structure

For the first part of the assignment, you should start by designing the BSTNode class for the BST. Create a class for the BSTNode data that will have as its members a character and a string. The character will hold the English text character, and the string will hold the corresponding Morse code characters for that English text character. You should also define left and right child pointers that point to BSTNode objects. You must have a constructor that accepts arguments to set the English text character and Morse code string.

2. (50 pts) Create the BST code and create a Morse *lookup* BST

Next, you should be able to read in the Morse table from a file called "MorseTable.txt". You should rearrange the Morse table in the file to make sure that your lookup tree is balanced. I recommend that you diagram a tree that provides a balanced tree so that you know how to order your "MorseTable.txt" file. Think about the order of insertions. However, the tree does not have to balance itself.

The tree should be built by the constructor for the BST. This means the constructor must open and read the file, create nodes for each character in the tree, insert the nodes into the tree (using an insert () function), and close the file. Note: the tree object could be declared as const, since all changes to it are being performed in the constructor. However, if you declare your object as a const, be sure to also declare your print () and search () functions as const. You should arrange the tree so that it is alphabetically ordered from left to right. Create a print ( ) function that uses recursion to traverse the tree in order (left most printed first). Also, build a search ( ) function that will return the Morse code string for each English character searched for. Do you need to return a found indicator from the search function? Should you use recursion? Finally, implement a destructor, which destroys the entire tree.

Morse Code Alphabet:

A .-	N -.	0 -----
B -...	O ---	1 .----
C -.-.	P .--.	2 ..---
D -..	Q --.-	3 ...--
E .	R .-.	4 ....-
F ..-.	S ...	5 .....
G --.	T -	6 -....
H	U ..-	7 --...
I ..	V ...-	8 ---..
J .---	W .--	9 ----.
K -.-	X -..-	FULLSTOP .-.-.-
L .-..	Y -.--	Comma ',' --..--
M --	Z --..	Query '?' ..--..

3. (30 pts) Putting the pieces together

First, print the current tree. Next, you must open a file called "Convert.txt", which consists of English alphabetic characters, spaces, commas, and periods. You must "look" for each English character with a search ( ) function on the BST, and print the Morse code string for that character. For each character in "Convert.txt", convert the character to a Morse code string. Each Morse character in the string will be separated by a space. Each complete Morse string will be separated by three spaces. Each newline character will be echoed to the screen. Note: you should convert any lowercase English characters to uppercase before processing the English text.

Below is an example test file (you should add more characters to test all conversions!):

(Convert.txt)
This is a test of the cpts 122
Morse code conversion tool.

(Echoed to screen)
- ....- - . ... - --- ..-. - .... . -.-. .--. - ... .---- ..--- ..---
-- --- .-. ... . -.-. --- -.. . -.-. --- -. ...- . .-. ... .. --- -. - --- --- .-.. .-.-.-

BONUS (15 pts): Implement a BSTNode and BST class template. Think about: how do you accommodate two different types in a class template?

IV. Submitting Assignments:

1. Using the OSBLE+ MS VS plugin, please submit your solution. Please visit http://osble.codeplex.com/wikipage?title=Submitting%20an%20Assignment&referringTitle=VS%20Plugin for more information about submitting using OSBLE+.
2. Your project must contain at least one header file (.h files) and two C++ source files (which must be .cpp files).
3. Your project must build properly. The most points an assignment can receive if it does not build properly is 65 out of 100.

V. Grading Guidelines:

This assignment is worth 100 points. Your assignment will be evaluated based on a successful compilation and adherence to the program requirements. We will grade according to the following criteria:

- 5 pts – Appropriate top-down design, style, and commenting according to class standards
- 15 pts - Defining the BSTNode structure
    1. 4 pts correct data members in node (char, string, left and right pointers)
    2. 3 pts correct constructor
    3. 8 pts other member functions
- 50 pts - Creating the BST code and create a Morse *lookup* BST
    1. 2 pts correct data member in BST (root)
    2. 10 pts correct insert ()
    3. 2 pts for opening "MorseTable.txt"
    4. 2 pts for closing "MorseTable.txt"
    5. 6 pts for reading contents of "MorseTable.txt"
    6. 5 pts for correct print ()
    7. 9 pts for correct search ()
    8. 6 pts for correct constructor
    9. 8 pts for correct destructor
- 30 pts - Putting the pieces together
    1. 5 pts for printing the tree
    2. 2 pts for opening "Convert.txt"
    3. 2 pts for closing "Convert.txt"
    4. 6 pts for reading contents of "Convert.txt"
    5. 10 pts for performing conversion of English to Morse code
    6. 5 pts for echoing Morse code to screen
- **BONUS 15 pts** – Working BSTNode and BST class template