**Approach to PROJ 2, BST ADT**

Riley Dorough / Chandler Whitley

27JAN20

This project was completed through the collaborative efforts of the ultra-genius Chandler Whitley & the innovative Riley Dorough.

**Doubly Threaded Binary Search Tree**

Requirements & Contemplation:

* First, create a complete *binary search tree*.

**BSTNode.h**

To attack this project, first map the dependency of the headers from the ground up and find the lowest level header that allows us to edit it. This should be the *ABag.h* header. This header should include our implementation of a *BagADT* and include *BagADT* related variables & methods.

DLBST variables:

* Key value of type *Key*
* Data value of type *E*
* Pointer to left child of type *BSTNode*
* Pointer to right child of type *BSTNode*
* Bool left and right flags for threads
  + Are implemented as bit fields with 1 singular bit

BSTNode class (construct & destruct):

* Zero Argument Constructor
  + Sets both pointers to NULL
  + Set flags to 0 (If 1: points to child, 0 points to in-order traversal or reverse in-order traversal). You can also override and specify a value.
* Custom Constructor
  + Assign the *Key* & *Value*
  + Set the pointers to NULL
  + Set flags to 0, or user input
  + Level variable equals 0 unless manually overwritten
* Destructor
  + Deletes left and right child pointers

BSTNode methods:

* Set, Get, & isLeaf methods already defined and written. Thank you.
* Added set and get methods for the flag data members. (Never used, due to threading)
* Added set and get methods for level variable.

**BST.h**

Handles the “management” of what methods to call on with the given *Key* & *E* values. This should help hide implementation from the end user. This is where we used the insert(), insertHelp(), printInorder(), and reversePrintInorder() methods.

BST Algorithm for Printing Non-Recursively:

* Verbally sudo-codded the algorithm then implemented into code (Far easier than trying to reexplain in English on here, lots of pictures)

BST variables:

* Integer for number of nodes.

BST class (construct & destruct):

* Constructor
  + Should use polymorphism to reinitialize a private data member to the size that is passed into the Dictionary constructor from the main .cpp file.
* Destructor
  + should delete any private data members.

**DTBSTMain.cpp**

1. Insert list of keys and values, according to instructions.

2. Call on the print function, then the in order traversal function, and finally the reverse in order traversal function.

3. High five each other.

4. Get 10% closer to that 4.0 Term GPA

**Coding Process**

Step 1 (day 1): Read, Learn, Prepare

* Study double-threaded binary trees in depth and learn the basics of their implementation
* Create project in Visual Studios
* Learn how to use debugger more efficiently

Step 2 (day 3): Warmups

* Start with BSTNode.h
* Add Boolean flags for double-threaded binary trees
  + Will switch to bit fields later
  + Two flags per node
* Create functions corresponding to the flags
* Each node has a right and left flag
* Pointer is regular if bit is a 1
* Pointer is a thread if bit is a 0
* Note: nodes can have one regular pointer (points to a child) and one thread pointer (points to inorder successor).

Step 3 (days 5-7): Inserthelp() method generation

* In pseudocode create basic algorithm with each case
* Define steps necessary for each case
  1. You are inserting the first Node
     + Only current node is the dummy
     + If tree is empty, both dummy pointers will point to itself
     + Make dummy point to the node you are inserting
  2. You are inserting a Node into a tree that already has at least one node
     + Must decided where to place this new node based on the data.
     + Must be done iteratively
       - If data is less than root . . . go left
       - If data is more than root . . . go right
       - Keep iterating until a node is found with the correct side (L or R depending on data) with flag 0 (Not pointing to Child)
       - Once found, reassign pointers and flags appropriately
* Check work using a test program

Step 4 (day 8): Create inorder traversal method using pseudocode first and then translate into code

Step 5 (day 9): Create reverseinorder traversal method. Draw out tree and think through the reverse traversal logically. Then use pseudocode and translate to actual code

Step 6 (day 10 – finish): Test everything and make any needed changes. Ensure all rules are followed and submit.