Bryce Hynes
COIS 3020H
March 14 MMXVII

Test: Print() Method + it's extension.
Purpose: The Print method is the most important since it displays the tree in a readable form.

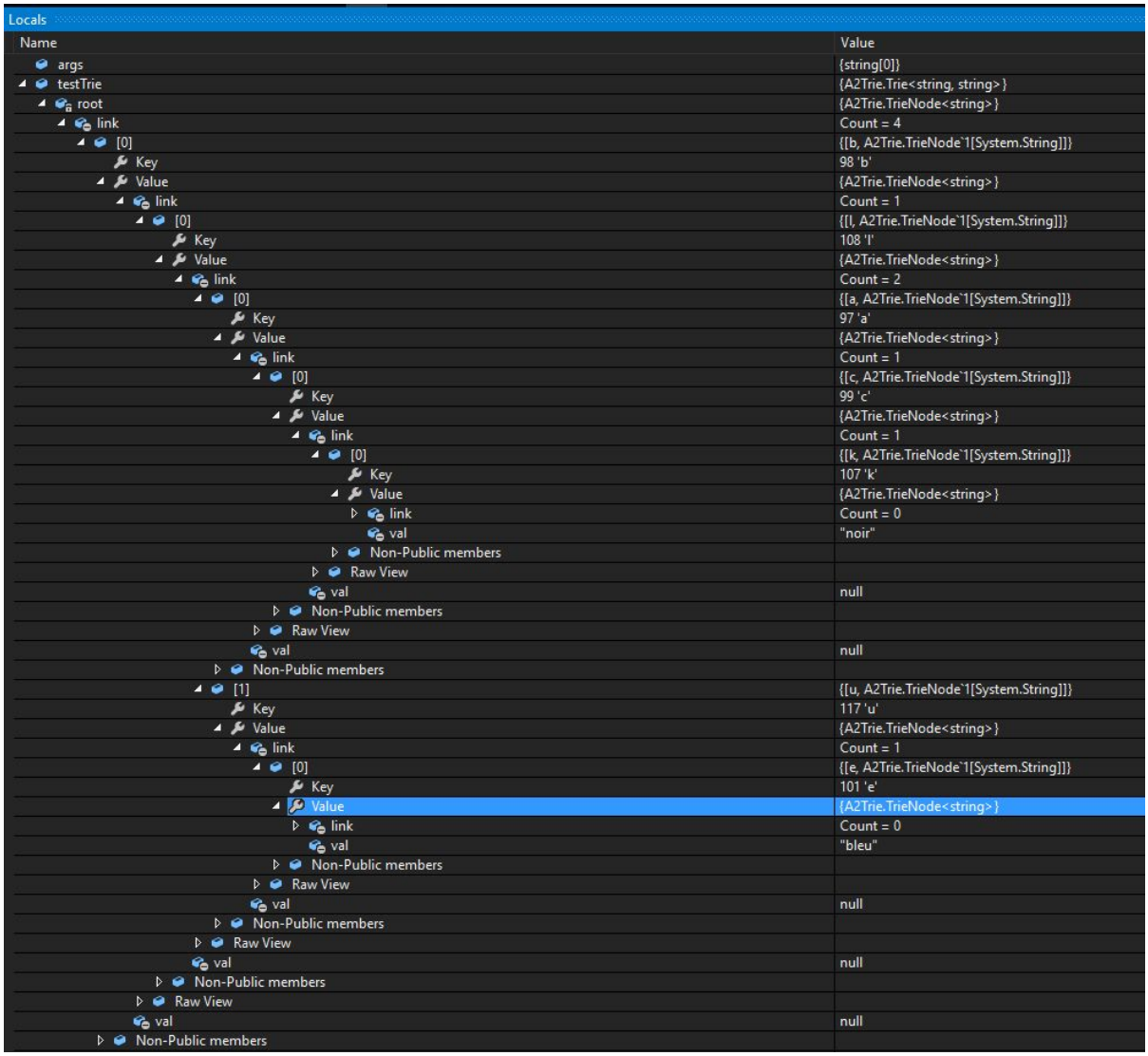
```
internal void Print()
{
    TrieNode<ValueType> curr = root;
    string engWord = ""; // using StringBuilder because of its ability to save the keys into a string
    Print(curr, engWord); // doing it this way since it's recursive and easier to traverse
}
/// <summary> Extension to Print()
void Print(TrieNode<ValueType> curr, string EnglishWord)
{
    StringBuilder engWord = new StringBuilder(EnglishWord);

    if (curr.val != null)
    {
        Console.WriteLine("{0} -- > {1}", engWord.ToString(), curr.val.ToString() ); //
    }

    for (char bruteForce = 'a'; bruteForce <= 'z'; bruteForce++)
    {
        if (curr.link.ContainsKey(bruteForce))
        {
            //curr = curr.link[bruteForce];
            Print( curr.link[bruteForce], ( engWord.Append(bruteForce).ToString() ) );
            //engWord.Append(bruteForce); // add the char that is present in the KeyLis

            ///TODO: get this method working
            /// the trick is int he order of th erecursion
            /// it's 3am
            /// it retians the most recent char from the previous print and carries tha
        }
    }
}
```

```
C:\WINDOWS\system32\cmd.exe
black -- > noir
blaue -- > bleu
bred -- > rouge
brwhite -- > blanc
brwyellow -- > jaune
Press any key to continue . . .
```

Result: the Method is supposed the traverse the trie in a depth-first manner to ensure the requirement of alphabetical order is achieved. While the output is less than satisfactory since it retains some of the letters from previous prints throughout, the order is still achieved somehow.

Bryce Hynes
COIS 3020H
March 14 MMXVII

Test: Add(KeyType k, ValueType v) method
Purpose: to verify that the Trie Was structured correctly\



Result: Using just the B Trie as an example of the structure. The Locals watcher of Visual Studio shows that when the branch occurs with a and u. Two new Tries are formed and correctly added. This is further demonstrated in the Translate and Remove functions, which rely on Add to function properly.

Bryce Hynes
COIS 3020H
March 14 MMXVII

Test: Valuetype Translate (string k) method
Purpose: Essentially this is the same as returning the value that is located in place of the end of string boolean that would normally exist in a Trie.

```
internal ValueType Translate(string k)
{
    Console.WriteLine("Translating {0}...", k);

    //used to navigate the tree
    TrieNode<ValueType> curr = root;
    string Key = k; // allows string to be changed into some other data type
    Key.ToUpper(); // this will add consistency in the tree, since chars are case sensitive

     //Loop that will traverse the Trie to the final letter of the word
    for (int chIndex = 0; chIndex < Key.Length; chIndex++)
    {
        // if the current node has the specified char in its dictionary, move down the Trie Tree
        if (curr.link.Keys.Contains(Key[chIndex]))
        {
            curr = curr.link[Key[chIndex]]; // curr becomes the child node

            //if
            if (chIndex == (Key.Length - 1))
            {
                // val is the Translation
                return curr.val;
            }
        }
        // if the letter isn't in the dictionary, then the word doesn't exist.
        else
        {
            Console.Write("Error - The word '{0}' does not exist ", k);
            return default(ValueType);
        }
    }//end loop

    // this is here so that all code paths return a value, but it will never make it here.
    Console.WriteLine("Error - Outside of for loop in [internal ValueType Translate(string k)] - D
    return default(ValueType);
}
```

```
E:\Dropbox\COIS3020\A1Q2\A2 - Trie\A
Translating black...
noir
Translating blue...
bleu
Translating red...
rouge
Translating white...
blanc
Translating yellow...
jaune
```

Result: The output is exactly what went into the dictionary, and it can be deemed a success

```
//test translate
Console.WriteLine(testTrie.Translate("black"));
Console.WriteLine(testTrie.Translate("blue"));
Console.WriteLine(testTrie.Translate("red"));
Console.WriteLine(testTrie.Translate("white"));
Console.WriteLine(testTrie.Translate("yellow"));
```

```
//test add
testTrie.Add("black", "noir");
testTrie.Add("blue", "bleu");
testTrie.Add("red", "rouge");
testTrie.Add("yellow", "jaune");
testTrie.Add("white", "blanc");
```

Bryce Hynes
COIS 3020H
March 14 MMXVII

Test: bool Remove(string k)
Purpose: To ensure that the removal of a work does not destroy other words that branch from it, and that the word is actually removed itself. To do this I will be testing if the word can be translated after removal, if another word can be retrieved that branched from the removed word, and if a word that doesn't exist can be removed

```
//test removal of word
testTrie.Remove("black");

//test translate of now - removed word
Console.WriteLine(testTrie.Translate("black"));

//test translate again of blue so that removing black didn't destroy the trie
Console.WriteLine(testTrie.Translate("blue"));

//test removal of word that never existed
testTrie.Remove("HelloWorld");
```

```
internal bool Remove(string k)
{
    //used to navigate the tree
    TrieNode<ValueType> curr = root;// insertion point for the trie
    TrieNode<ValueType> prevCurr; // used for backtracking the Trie
    string Key = k; // allows string to be changed into some other data type
    Key.ToUpper(); // this will add consistency in the tree, since chars are case sensitive
    int chIndex = 0;

    while (curr.link.Count != 0 && chIndex < Key.Length)// a for loop wouldn't cut it in this scenario
    {
        prevCurr = curr;

        // this is the first time I've used a try..catch
        try
        {
            curr = curr.link[Key[chIndex]]; // was crashi
        }
        catch(KeyNotFoundException knf)
        {
            //Console.WriteLine(knf); // error output
            return true;
        }

        //if no links found, erase this node along the wa
        if (curr.link.Count == 0 )
        {
            curr = prevCurr; // since the program needs to back up one node to remove this one.
            Console.WriteLine("Removing '{0}' at index {2} from '{1}'", Key[chIndex], Key, Convert.ToString(chIndex));
            curr.link.Remove(Key[chIndex]); // remove the node with the keyt being the current char at chIndex of k
            Remove(k); // RECURSION
            return true;
        }
        chIndex++;
    }
    return false;
}
```

```
E:\Dropbox\COIS3020\A1Q2\A2 - Trie\A2 - Trie\bin\Debug\A2 - Trie.e
Removing 'k' at index 4 from 'black'
Removing 'c' at index 3 from 'black'
Removing 'a' at index 2 from 'black'
Translating black...
Error - The word 'black' does not exist
Translating blue...
bleu
```

Result: The Remove Function is working as advertised