BST Iteration WSKey

The project **BSTIterator** has code to test functions to convert BST to a vector as well as to test the implementation of a BST iterator. All the test code works on the tree shown to the right.

**toVector**

```cpp
void vecBuilder(BSTNode< char >* curNode, vector< char >&
    if (curNode == nullptr)
        return;
    // in order traversal: left, self, right
    vecBuilder(curNode->left, theVec);
    // don't print current, add to vector
    theVec.push_back(curNode->value);
    vecBuilder(curNode->right, theVec);
}
```

**Iterator**

CharBST already has code to create begin and end iterators. The **CharBSTIterator** class has all the necessary functions declared and built with two exceptions: constructor and preincrement.

Since we are using the front of a deque as a stack:
To push: `positionStack.push_front(node*)`
To pop: `node* = positionStack.pop_front()`
To get top (without removing): `node* = positionStack.front()`

**Constructor**

```cpp
// Constructor - takes node to use as root
// nullptr signifies that this should be an end iterator
CharBSTIterator::CharBSTIterator(BSTNode< char >* root)
{
    // nullptr is the last thing we should see - marks the end of the traversal
    positionStack.push_front(nullptr);
    // assuming we have a valid starting point
    if (root != nullptr) {
        // TODO - CharBSTIterator constructor
        BSTNode< char >* temp = root;  // where we are in tree
        while (temp != nullptr) {
            // until we run out of nodes
            positionStack.push_front(temp);  // push current node,
            temp = temp->left;  // slide left
        }
        // note - could just make the surrounding if the while loop
    }
}
```

`++()`
CharBSTIterator CharBSTIterator::operator++()
{
    // TODO - CharBSTIterator ++ operator
    BSTNode<char>* temp = positionStack.front();
    positionStack.pop_front();
    temp = temp->right;
    while (temp != nullptr) {
        positionStack.push_front(temp);
        temp = temp->left;
    }

    return *this;
}