Conrad Horton
CSC382 - September 2016
9-26-2016
Binary Search Tree

My Binary Search Tree implements a parent/child node structure where each parent node has up to two child nodes (left and right). Each node in the tree has a key value and some sort of data value. The size of the key determines which side it goes on; if the value is larger than the parent node, it goes on the right side or else it goes to the left. The node will keep going down the tree until it becomes an external node--the last node in that branch. It provides the user the ability to add nodes, delete nodes, and find the max key value.

Binary search trees are very useful for finding data quickly since its execution time at worst is O(logn) for accessing data. Since this tree lacks any sort of accessing it is somewhat useless except for taking up space. It could be expanded upon and be pretty cool.

BST() - Creates a new instance with a root node set to *nullptr.*

add(long long key, T t) - Appends a node with the passed in *key* and and data object of type T.

del(long long key) - Searches the tree for a no de with a key value of *key* and removes it. The subtree is then reattached to the main tree.

max() - Traverses the tree and finds the maximum key value.

Sample usage:

```
#include <iostream>
#include <string>
#include <random>
#include <ctime>
#include "BST.h"

int main()
  {
  BST<int> bst;

  long long key    = 0;
  long long delKey = 0;
  /** Load array with 100 key value pairs. */
  srand (time(NULL));

  for(int i = 0; i < 100; i++)
    {
    key = rand() % 1000 + 1;
    delKey = key < 500 ? key : 0;

    bst.add(key, i);
    }

  bst.max();
  bst.del(delKey);
  bst.max();
  //system("pause");
  return 0;
  }
```