# James Hudspeth’s Binary Search Tree

This is a binary search tree built in C++. It is a linked data structure that consists of data stored in nodes that are related by the node’s value. Each node contains three pointers, linking each node to the left, right, and parent node. If the tree is empty, then a root node will be created for the first value. The next node’s value will be compared with the root’s value; if lower, the new node will be inserted on the left. If the new value is higher, it will be inserted to the right. This pattern is followed throughout the tree, moving down the tree if a node is already occupied. The program itself makes use of a simple user input menu. After each command the menu will reappear and ask the user what they wish to do next. The user enters commands in the form of a single digit corresponding to each menu option. The binary search tree class is templated, allowing it to work with multiple different types of data, but the tree created is currently set to int.

# Member Variables

Node

element – data stored (public)

left – pointer to left node (public)

right – pointer to right node (public)

parent – pointer to the parent node (public)

BSTree

root – pointer to the root node (private)

Main

menu – char variable to hold user input for menu options

intTree – binary search tree for integers

testTree – pointer binary search tree for testing purposes

value – user input to pass to list functions

foo – junk variable, used to end program

# Functions/Methods

isEmpty – simple function to check if the list is currently empty of data.

Parameters – none

Return – true if list is empty, false if list contains data

Insert – function to insert data to the list.

Parameters – (const T &elem) value to be inserted into the list, defined in BinarySearchTree as an alias of its first template parameter (T).

Return – none

Usage – used to add the root value to the tree, automatically calls overloaded version if root already contains data.

Insert – overloaded function to insert data to the list after the root.

Parameters – (Node<T> \*curr, const T &elem)

Node<T> \*curr – pointer to the current location within the tree

Const T &elem – value to be inserted into the list, defined in BinarySearchTree as an alias of its first template parameter (T).

Return – none

Usage – used to add values after root. Is automatically called by the initial Insert function call if root already contains data.

Delete – function to delete data from the list.

Parameters – (Node<T> \*r) address of node to be deleted

Return – true if the value is deleted, false if not

KillTree – clears the whole tree of data (not entirely implemented)

Parameters – (Node<T> \*curr) address of node to start clearing tree from

Return – none

ViewTree – displays the tree’s contents in sequential order.

Parameters – none

Return – none

Usage – used to display the entire tree, calls overloaded display.

ViewTree – overloaded function to display the tree’s contents in sequential order.

Parameters – (Node<T> \*curr) – address of node to start displaying tree from

Return – none

Usage – called automatically from initial ViewTree function call

FindMax – searches the entire tree for the maximum value.

Parameters – none

Return – address of the maximum value within the entire tree

Usage – used to find the maximum value within the entire tree, automatically calls overloaded function with root as the parameter.

FindMax – overloaded function to search for the maximum value of the tree

Parameters – (Node<T> \*curr) address of node to start searching from

Return – address of the maximum node

Usage – called automatically from initial FindMax function call. Can be used to find the maximum value in a branch.

Find – searches the tree for a value and returns the address.

Parameters – (T value) value to be found within the list, defined in BinarySearchTree as an alias of its first template parameter (T).

Return – returns the address of the node if found

Usage – used to find the address for deletion, can be used to look up the address of any node