# CS 300 Binary Search Tree Reflection and Pseudocode

Robert Umland

Robert.Umland@SNHU.EDU

Southern New Hampshire University

**Code Reflection**

This week’s project was to take a .csv file of eBid auction sales, read them into the program and hold them in a Binary Search Tree. This could then be searched to return a pre-determined line. That line could be read out and removed by choices in the existing menu. We were given an unfinished code that needed to be completed for the above to work properly. I had two main challenges with this project. First was the extent of the FIXME comments. There were many parts that needed to be incorporated. But that wasn’t too hard. The comments and the ZyBooks pretty much made the code easy to write. There was one stand out, though. I did not understand how to write the destructor. After some help, I worked out a helper method that seems to work properly. The other was simple but incredibly frustrating. At one point in my work, nothing launched, it all was failing, and I was struggling. Turned out I had accidentally deleted one of the “/” at the very beginning that made the box around where our name goes. An hour of hair pulling for such a little error.

**Pseudocode**

// Forward declarations

function strToDouble(string str, char ch) -> double

// Define a structure to hold bid information

structure Bid

string bidId // unique identifier

string title

string fund

double amount

constructor Bid()

amount = 0.0

// Internal structure for tree node

structure Node

Bid bid

Node\* left

Node\* right

// Default constructor

constructor Node()

left = nullptr

right = nullptr

// Initialize with a bid

constructor Node(Bid aBid) : Node()

bid = aBid

// Binary Search Tree class definition

class BinarySearchTree

private:

Node\* root

function addNode(Node\* node, Bid bid)

function inOrder(Node\* node)

function postOrder(Node\* node)

function preOrder(Node\* node)

function removeNode(Node\* node, string bidId) -> Node\*

public:

constructor BinarySearchTree()

destructor ~BinarySearchTree()

function InOrder()

function PostOrder()

function PreOrder()

function Insert(Bid bid)

function Remove(string bidId)

function Search(string bidId) -> Bid

function IsEmpty() -> bool

return root == nullptr

constructor BinarySearchTree()

root = nullptr

function destroyNode(Node\* node)

if node != nullptr

destroyNode(node->left)

destroyNode(node->right)

delete node

destructor ~BinarySearchTree()

destroyNode(root)

function InOrder()

inOrder(root)

function PostOrder()

postOrder(root)

function PreOrder()

preOrder(root)

function Insert(Bid bid)

if root == nullptr

root = new Node(bid)

else

addNode(root, bid)

function Remove(string bidId)

root = removeNode(root, bidId)

function Search(string bidId) -> Bid

Node\* current = root

while current != nullptr && current->bid.bidId != bidId

if bidId < current->bid.bidId

current = current->left

else

current = current->right

if current != nullptr && current->bid.bidId == bidId

return current->bid

Bid bid

return bid

function addNode(Node\* node, Bid bid)

if bid.bidId < node->bid.bidId

if node->left == nullptr

node->left = new Node(bid)

else

addNode(node->left, bid)

else

if node->right == nullptr

node->right = new Node(bid)

else

addNode(node->right, bid)

function inOrder(Node\* node)

if node != nullptr

inOrder(node->left)

print node->bid.bidId + " | " + node->bid.title + " | " + node->bid.amount + " | " + node->bid.fund

inOrder(node->right)

function postOrder(Node\* node)

if node != nullptr

postOrder(node->left)

postOrder(node->right)

print node->bid.bidId + " | " + node->bid.title + " | " + node->bid.amount + " | " + node->bid.fund

function preOrder(Node\* node)

if node != nullptr

print node->bid.bidId + " | " + node->bid.title + " | " + node->bid.amount + " | " + node->bid.fund

preOrder(node->left)

preOrder(node->right)

function removeNode(Node\* node, string bidId) -> Node\*

if node == nullptr

return node

if bidId.compare(node->bid.bidId) < 0

node->left = removeNode(node->left, bidId)

else if bidId.compare(node->bid.bidId) > 0

node->right = removeNode(node->right, bidId)

else

if node->left == nullptr && node->right == nullptr

delete node

node = nullptr

else if node->left != nullptr && node->right == nullptr

Node\* temp = node

node = node->left

delete temp

temp = nullptr

else if node->right != nullptr && node->left == nullptr

Node\* temp = node

node = node->right

delete temp

temp = nullptr

else

Node\* temp = node->right

while temp->left != nullptr

temp = temp->left

node->bid = temp->bid

node->right = removeNode(node->right, temp->bid.bidId)

return node

function displayBid(Bid bid)

print bid.bidId + ": " + bid.title + " | " + bid.amount + " | " + bid.fund

function loadBids(string csvPath, BinarySearchTree\* bst)

print "Loading CSV file " + csvPath

csv::Parser file = csv::Parser(csvPath)

vector<string> header = file.getHeader()

for each c in header

print c + " | "

print ""

try

for i from 0 to file.rowCount() - 1

Bid bid

bid.bidId = file[i][1]

bid.title = file[i][0]

bid.fund = file[i][8]

bid.amount = strToDouble(file[i][4], '$')

print "Item: " + bid.title + ", Fund: " + bid.fund + ", Amount: " + bid.amount

bst.Insert(bid)

catch csv::Error e

print e.what()

function strToDouble(string str, char ch) -> double

str.erase(remove(str.begin(), str.end(), ch), str.end())

return atof(str.c\_str())

function main(int argc, char\* argv[])

string csvPath, bidKey

switch argc

case 2

csvPath = argv[1]

bidKey = "98223"

case 3

csvPath = argv[1]

bidKey = argv[2]

default

csvPath = "CS 300 eBid\_Monthly\_Sales\_Dec\_2016.csv"

bidKey = "98223"

clock\_t ticks

BinarySearchTree\* bst = new BinarySearchTree()

Bid bid

int choice = 0

while choice != 9

print "Menu:"

print " 1. Load Bids"

print " 2. Display All Bids"

print " 3. Find Bid"

print " 4. Remove Bid"

print " 9. Exit"

print "Enter choice: "

cin >> choice

switch choice

case 1:

ticks = clock()

loadBids(csvPath, bst)

ticks = clock() - ticks

print "time: " + ticks + " clock ticks"

print "time: " + ticks \* 1.0 / CLOCKS\_PER\_SEC + " seconds"

if bst.IsEmpty()

print "No bids were loaded."

else

print "Bids loaded successfully."

case 2:

bst.InOrder()

case 3:

ticks = clock()

bid = bst.Search(bidKey)

ticks = clock() - ticks

if not bid.bidId.empty()

displayBid(bid)

else

print "Bid Id " + bidKey + " not found."

print "time: " + ticks + " clock ticks"

print "time: " + ticks \* 1.0 / CLOCKS\_PER\_SEC + " seconds"

case 4:

bst.Remove(bidKey)

print "Good bye."

return 0