The binary tree is a great way to organize and search for data. It consists of levels through a list. The binary tree is sort of like a linked list, however, a node may have up to two ‘next’ nodes. This cuts searching for data by quite a lot. Creating the program for the bid wasn’t difficult, compared to the previous assignments. I will say that I do think the previous assignments have helped me quite a lot for this one, as without them, I probably would have had a more difficult time. The only roadblock that I encountered was with the provided code. The strtodouble function wasn’t working and I figured out, by using the link provided, a few #includes were missing. Once added, the program worked fine.

**Fixme 1 – Internal Struct**

DEFINE Bid bid

DEFINE Pointer Node right

DEFINE Pointer Node left

CONSTRUCTOR Node w/no param {

SET right to nullptr

SET left to nullptr

}

CONSTRUCTOR Node w/Bid aBid {

SET bid to aBid

}

**Fixme 2a – Insert**

IF root is 0

SET root to new Node w/bid

ELSE

CALL addNode with root and bid param

**Fixme 2b – Add Node**

IF node bid bidId compare to bid bidId is great then 0 {

IF node left is nullptr

SET node left to new node w/bid

ELSE

CALL addNode with node left and bid

}

ELSE {

IF node right is nullptr

SET node right to new node w/bid

ELSE

CALL addNode w/ node right and bid

}

**Fixme 3 – Search**

DEFINE Pointer Node current

SET current to root

WHILE current is not nullptr {

IF current bid bidid compare bidid is 0

RETURN current bid

IF bid id compare current bid bidid is less than 0

SET current to current left

ELSE IF bid id compare current bid bidid is greater than 0

SET current to current right

}

DEFINE Bid bid

RETURN bid

**Fixme 4a – Remove**

FUNC Remove w/ string bidid

CALL removeNode w/root and bidid

**Fixme 4b – Remove Node**

FUNC Node pointer removeNode w/Node pointer node and string bidId params {

IF node is nullptr

RETURN node

IF bidid compare node bid bidid is less than 0

SET node left to return value of removenode w/node left and bidId

ELSE IF bidid compare node bid bidid is greater than 0

SET node right to return value of removenode w/node right and bidid

ELSE {

IF node left is nullptr and node right is nullptr {

DELETE node

SET node to nullptr

}

ELSE IF node left IS NOT nullptr AND node right IS nullptr {

DEFINE Pointer Node tempNode

SET tempNode to node right

DELETE tempNode

}

ELSE IF node left IS nullptr AND node right IS Not nullptr {

DEFINE Pointer Node tempNode

SET tempNode to node left

DELETE tempNode

}

ELSE {

DEFINE Pointer Node tempNode

SET tempNode to node right

WHILE tempNode left is not nullptr

SET tempNode to tempNode left

SET node bid to tempNode bid

SET node right to return value of removeNode w/node right and tempNode bid bidId

}

}

RETURN node

}