1. Default Constructor

BinarySearchTree ()

INPUT: None

OUTPUT: None

root = nullptr

1. Destructor

BinarySearchTree::~BinarySearchTree()

INPUT: None

OUTPUT: None

WHILE (root does not equal nullptr)

Root = removeNode

1. InOrder Public function

BinarySearchTree::InOrder()

INPUT: None

OUTPUT: None

Call inOrder Private Function

1. PostOrder Public function

BinarySearchTree::PostOrder()

INPUT: None

OUTPUT: None

Call postOrder Private Function

1. PreOrder Public function

BinarySearchTree::PreOrder()

INPUT: None

OUTPUT: None

Call preOrder Private Function

1. Search

Bid BinarySearchTree::Search(string bidId)

INPUT: None

OUTPUT: None

INTRODUCE Bid bid

WHILE (current does not equal nullptr)

IF (current’s bidId equals bidId)

RETURN current’s bid

ELSE IF ( BidId is less than current’s bidId )

Traverse LEFT

ELSE

Traverse Right

END WHILE

Return bid

1. Add Node

BinarySearchTree::addNode(Node\* node, Bid bid)

INPUT: None

OUTPUT: None

IF (Bid’s bidId is smaller than Node’s bidId)

IF(No Left Node)

Node’s bidId becomes left

ELSE

Recourse down the left node

ELSE IF (Bid’s bidId is greater than Node’s bidId)

IF(No right Node)

Node’s bidId becomes right

ELSE

Recourse down the right node

1. Private inOrder

BinarySearchTree::inOrder(Node\* node)

INPUT: None

OUTPUT: bidID, title, amount, fund

IF (Node does not equal nullptr)

inOrder is not left

Output bidID, title, amount, fund

inOrder is right

1. Private postOrder

BinarySearchTree::postOrder(Node\* node)

INPUT: None

OUTPUT: bidID, title, amount, fund

IF (Node does not equal nullptr)

PostOrder(left)

PostOrder(right)

Output bidID, title, amount, fund

1. Private preorder

BinarySearchTree::preOrder(Node\* node)

INPUT: None

OUTPUT: bidID, title, amount, fund

IF (Node does not equal nullptr)

Output bidID, title, amount, fund

preOrder (left)

preOrder (right)

1. Remove Node

INPUT: None

OUTPUT: None

IF (node equals nullptr)

RETURN node

IF (bidId is less than Node’s bidID)

Remove left node

ELSE IF (bidId is greater than Node’s bidID)

Remove right Node

ELSE

IF (The left and right pointers equal nullptr)

DELETE node

ELSE IF (The left node does not equal nullptr but the right one does equal nullptr)

Delete Node

ELSE IF (The left node equals nullptr but the right one does not equal nullptr)

Delete Node

Node of temp = right node

WHILE (left node is not nullptr)

keep moving temp left

END WHILE

Nodes’s bid equals temp’s bid

Return node

1. Load Bids

DISPLAY “Loading CSV file”

INITIALIZE the CSV Parser using the given path

READ and DISPLAY header row

TRY

FOR (All Rows of the CSV file)

CREATE data Structure

bid

bidID

title

fund

amount

END CREATE

PUSH bid to the end of the hash table

END FOR

CATCH

ERROR e

1. String to Double

INPUT: Strings

OUTPUT: Doubles

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

return atof(str.c\_str());

1. Main

Display Menu

"Menu:"

" 1. Load Bids"

" 2. Display All Bids"

" 3. Find Bid"

" 4. Remove Bid"

" 9. Exit"

"Enter choice:

Case 1

Initialize a timer variable before loading bids

Complete the method call to load the bids

Calculate elapsed time and display result

Case 2

PrintALL function

Case 3

Ticks equals clock function

IF (bidID is empty)  
 displayBid function

END IF

ELSE

DISPLAY "Bid Id " << bidKey << " not found."

END ELSE

DISPLAY ticks

DISPLAY Clocks per Second

Case 4

Remove function

1. Exit

DISPLAY “Goodbye”

TERMINATE Program