**Binary Tree:**

Adding a record

Parameters — new node

Return — nothing

*Create a new node and add it to the tree*

Insert(keyNode : node pointer)

Recursive call to root

Insert(parent: node pointer, keyNode : node pointer)

If (parent is null) *{Base Case 1}*

Root := keyNode

If (keyNode->data < parent->data)

If (parent->left is null) *{Base Case 2}*

parent->left := keyNode

Else

Recursive call to parent->left

Else

If (parent->right is null) *{Base Case 3}*

parent->right := keyNode

Else

Recursive call to parent->right

Deleting a record

Parameters — data value

Return — boolean

*User will enter a game name. Delete the record from the tree*

remove(key: data value)

Target := search(key)

Parent := getParent(Target)

Recursive call to parent and target

remove(parent: node pointer, target: node pointer)

If (target is false) *{Case 1}*

Return false

If (target->left and target->right are not null) *{Case 2}*

Successor := target->right

SuccessorParent := target

While (Successor->left is not null)

SuccessorParent := successor

Successor := successor->left

Target->data := successor->data

Recursive call to successorParent and Successor

Return true

Else If (target is root) *{Case 3}*

If (target->left is not null)

Root := target->left

Else

Root := target->right

Else if (target-> left is not null) *{Case 4}*

If (parent->left = target)

parent ->left = target->left

Else

parent ->right = target->left

Else

If (parent->left = target) *{Case 4}*

parent ->left = target->right

Else

parent ->right = target->right

target->left = null

target->right = null

Return true

Searching for a record

Parameters — data value

Return — node pointer

*Search through the tree and display the*

*data for any node that contains the search data.*

Search(key: data value)

Return (recursive call to root of list)

Search(pos, node pointer key: data value)

If (node is not null)

If (key is equal to pos->data) *{Base Case}*

Return pos

Else if (key < pos->data)

Return (recursive call to node left of pos)

Else

Return (recursive call to node right of pos)

Return null *{returns null if node is not found}*

Get parent node

Parameters — node pointer

Return — node pointer

*Finds the parent node of a given node*

GetParent(target: node pointer)

If (target = root)

Return null

Return (recursive call to root)

GetParent(subRoot: node pointer, target: node pointer) *{Base Case 1}*

If (subRoot is null)

Return null

If (subRoot->left or subRoot->right is equal to target) *{Base Case 2}*

Return subRoot

If (target->data < subRoot->data)

Return (recursive call to subRoot->left)

Else

Return (recursive call to subRoot->right)

Display records in order

Parameters — boolean

Return — nothing

*Display data in ascending or descending order via inorder traversal.*

DisplayInOrder(isDescending: boolean) *{default value for isDescending is false}*

Recursive call to root

DisplayInOrder(pos: node pointer, isDescending: boolean)

If (isDescending is true)

If (pos is null)

Return

Recursive call to pos -> right

Print pos

Recursive call to pos -> left

Else

If (pos is null)

Return

Recursive call to pos -> left

Print pos

Recursive call to pos -> right

Display records breadth-first traversal

Parameters — output stream

Return — nothing

*print tree one level at a time to stream*

DisplayBreadthFirst (out: output stream)

For i = 1 to height()

Recursive call to i

DisplayBreadthFirst (out: output stream, level: integer, pos: node pointer) {pos default to root}

If (root is null)

Return

If (pos is root)

Print node to out

Else if (level > 1)

Recursive call to level and pos->left

Recursive call to level and pos->right

Get height

Parameters — nothing

Return — integer

*Finds the height of the tree by counting the right branch length versus the left*

height(pos: node pointer) {pos default to root}

If (pos is null)

Return 0

Else

Left height := (recursive call to pos->left)

Right height := (recursive call to pos->left)

If (Left height > Right height)

Return left height

Else

Return right height

**Main:**

Main:

Prompt user for file names

Open all files

If (input file is open)

Read input file

Else

Print error message

If (command file is open)

Read commands file

Else

Print error message

If (cidercade.dat is open)

Print DisplayBreadthFirst to file

Else

Print error message

Close all files

Editing a record

Parameters — String, integer, templated type

Return — nothing

*With a given game name, the program will update the record and confirm the change by*

*displaying the new record on the screen. The following items can be edited:*

*1. High score*

*2. Initials*

*3. Number of plays*

*▪ If number of plays is edited, the revenue should be recalculated*

*• $0.25 per play*

updateRecord(name: string, field: integer, value: string)

Node = search(name)

Switch (field)

Case 1:

Node->data->set high score to value

Case 2:

Node->data->set initials to value

Case 3:

Node->data->set number of plays to (string to integer of value)

*{update revenue built into set number of plays}*

Print Node

Read input file

Parameters — input stream

Return — nothing

*Read each line of the input file line by line and add all records to the binary tree*

readInputLine(in: input stream)

While (can in.get line into a temp string)

Create string stream from temp

temp.Get line into temp2 with comma delimiter

Name := temp2

temp.Get line into High\_score with comma delimiter

temp.Get line into Initials with comma delimiter

temp.Get line into temp2 with comma delimiter

Plays := string to integer of temp2

temp.Get line into temp2

Revenue := string to double of temp2

Create game with name, High\_score, Initials, Plays, Revenue

Create node with game

Insert node into list

Read commands file

Parameters — input stream

Return — nothing

*Reads and executes commands from commands file*

readCommandsFile(in: input stream)

While (can in.get line into a temp string)

Create string stream from temp

Temp >> field

Switch (field)

Case 1:

Find first and last quotes

Name := Substring using quotes

temp.Get line into High\_score with comma delimiter

temp.Get line into Initials with comma delimiter

temp.Get line into temp2 with comma delimiter

Plays := string to integer of temp2

temp.Get line into temp2

Revenue := string to double of temp2

Create game with name, High\_score, Initials, Plays, Revenue

Create node with game

Insert node into list

Print RECORD ADDED

Print game

Case 2:

temp.Get line into search term

Search for search term

If search returns true

Print <name> FOUND

Print node data

Else

Print <name> NOT FOUND

Case 3:

Find first and last quotes

Name := Substring using quotes

Temp >> field2 >> value

update Record with Name Field and value

Print <name> UPDATED

Print UPDATE TO <field> - VALUE <value>

Case 4:

temp.Get line into name

Print RECORD DELETED

Print (search name)

remove name

Case 5:

Temp >> sort

If (sort = “asc”)

displayInOrder(false)

Else if (sort = “des”)

displayInOrder(true)

Print two empty lines