Project Two Pseudocode

Andy Nguyen

CS – 2336.003

Smith

Main Class

Main

- **Ask Input File Name**
- Initialize and declare File, InputFileStream, and Scanner object
- Create a LinkedList playerList
- If the input file from user exists go ahead
    - While scanner object has a next line
        - **Read File Line** // parsing of the stat will be done within **player object**
- Else
    - Tell the user that the file could not be opened
- With the LinkedList playerList **Check for Multiple Entries**
- Then **Sort Players** if LinkedList playerList
- **Display Players Recursively**

- Create new a bunch of linked lists which store a stat from greatest to least if it makes sense for that stat **Sort Players by Stat (Greatest to Least)**
- Create a linked lists which store a stat (specifically the strike out stat) from least to greatest if it makes sense for that stat **Sort Players by Stat (Least to Greatest)**
- **Output Leaders** using parameters of **Find Leaders** (**Find Leaders** takes in the linked list sorted stats created in main) and a string describing the stat to be processed // this function is repeated for each stat that needs to find leader

Ask Input File Name

Parameters: Nothing

Return: string containing the file name

- Call a scanner object
- Ask for a file to be inputted by the user and store input into string

Read File Line

Parameters: scanner object, linked list of players

Return: player object

- Call strings that will take in the file line, the player name, and the player stats
- Use scanner object to scan in a line (scnr.nextLine) from the text file
- If file line scanned length is not equal to zero then go ahead // skips blank lines

    - For 0 : file line scanned length
        - If file line scanned character is equal to space
            - Take index of string from 0 to the space and store into player name string
            - Take the rest of string after the space and store it into player stat string
- Store the name and stats of the player in the player linked list// **player object** will handle the parsing of the stats string | playerList.append(new Player(<Name and stats>));

Link List Class

- Call a Node head which act as the starting point for our list

LinkedList Append

Parameters: Player object

Return: Nothing

- Call for a new node to be created and store player object which contains data within node
- If head node is equal to null
    - Set head node equal to new node
- Else if the list is not empty
    - Traverse the node list using while loop until it reaches the last node where next node is equal to null
    - Set the last node equal to the new node

LinkedList Delete

Parameters: String which holds a player's name

Return: Nothing

- Call a temporary node set equal to head and a previous node set equal to null
- If temporary node is not equal to null and player contained matches the name of the string parameter // checks if head matches the data that need to be deleted
    - Set the head node equal to temp's next node deleting it from list
    - Return to get out of function
- While temporary node is not equal to null and the temporary node's player's name is not equal to string parameter
    - Set previous node equal to temporary node
    - Set temporary node equal to next node
- If the temporary node is equal to null
    - Return to get out of function cause the string player name was not found in linked list
- Set the previous next node equal to the temporary next node to unlink from linked list

LinkedList Check for Multiple Entries

Parameters: Nothing

Return: Nothing

- Create a search node to use to check for multiple instances of that node set equal to head node
- Create a current node to traverse through linked list set equal to head node

- While search node is not equal to null
    - While current node is not equal to null
        - If current node's player's name is equal to search node's player's name and they are not the same node
            - Get stats of integer member variables from the search node and uses special setter methods which add on to the member variables of the current node, add the stats of search node and add them on to current node
            - **Delete** using search nodes string player name, deleting the first instance of the player
        - Set current node equal to next node of current node
    - Set current node equal to head node
    - Set search node equal to next node of search node

LinkedList Check List in Order

Parameters: Nothing

Return: Boolean value of whether linked list is in order

- Create a current node set equal to head node to traverse linked list
- While current node is not equal to null and next node to current is not equal to null
    - Create two strings which hold the player names of the current node and the next node
    - Create integer compareValue which stores the value returned from the "compareTo" function
    - If compareValue is greater than 0
        - That means the linked list is not in alphabetical order so return false
    - Set current node equal to the node next to the current node
- Return true cause nodes of linked list are in alphabetical order

LinkedList Sort Players

Parameters: Nothing

Return: Nothing

- Create a current node to help us traverse linked list
- While **Check List in Order** is equal to false
    - If the current node is not equal to null and the get the next node of current node is not equal to null
        - Call string which stores player name at current node
        - Call string which stores player name at the next node of current node

        - Call an int compare value which stores value of “compareTo” function // compareTo helps us determine whether two strings are in alphabetical order
        - If compare value is greater than 0, meaning the current node string is greater than the string of the next node
            - **Append** the current node // creates a copy of current node and put it at the end
            - **Delete** the first instance of the current node
        - Set current node equal to the node next to current node
    - If the node next to current node is equal to null // restart the traversal from the begging of the list to check if in order
        - Set current node equal to head node

LinkedList Display Players Recursively

Parameters: Node called current node used to help traverse

Return: Nothing

- If current node is not equal to null
    - Print out the player contained within the node utilizing the **To String** from Player class
    - Call the **Display Players Recursively** with the parameter being the next node of current node
- Use return to break out of recursive function
- **LinkedList Sort Players**

LinkedList Check List in Order by Stat (Greatest to Least)

Parameters: Copied list of the alphabetized linked list, String that represents desire stat to be checked

Return: Boolean value of whether the copied list for stats is from greatest to least

- Create a current node set equal to head node to traverse linked list
- While current node is not equal to null and next node to current is not equal to null // trying to check if the sort properly put stats from greatest to least
    - Create two integers which hold the **Which Stat to Get** stat of current node and node next to current
    - If stat of current node is less than stat of node next to current node
        - Return false meaning the list is not from greatest to least
    - Set the current node equal to the node next to the current node
- Return true cause list is from greatest to least

LinkedList Sort Players by Stat (Greatest to Least)

Parameters: String that represents desire stat to be checked

Return: Copied linked list

- Create a copied list of the alphabetized linked list

- Create a current node to help us traverse linked list set equal to head node of copied linked list
- While **Check List in Order by Stat (Greatest to Least)** is equal to false
    - If the current node is not equal to null and the get the next node of current node is not equal to null
        - Calls two integers which hold the **Which Stat to Get** stat of current node and node next to current
        - If stat of current node is less than stat of node next to current node
            - **Append** the current node in coped linked list // creates a copy of current node and put it at the end
            - **Delete** the first instance of the current node in copied linked list
        - Set current node equal to the node next to current node
    - If the node next to current node is equal to null // restart the traversal from the begging of the list to check if in order
        - Set the current node equal to head node of copied linked list

LinkedList Check List in Order by Stat (Least to Greatest)

Parameters: Copied list of the alphabetized linked list, String that represents desire stat to be checked

Return: Boolean value of whether the copied list for stats is from greatest to least

- Create a current node set equal to head node to traverse linked list
- While current node is not equal to null and next node to current is not equal to null // trying to check if the sort properly put stats from greatest to least
    - Create two integers which hold the **Which Stat to Get** stat of current node and node next to current
    - If stat of current node is more than stat of node next to current node
        - Return false meaning the list is not from greatest to least
    - Set the current node equal to the node next to the current node
- Return true cause list is from greatest to least

LinkedList Sort Players by Stat (Least to Greatest)

Parameters: String that represents desire stat to be checked

Return: Copied linked list

- Create a copied list of the alphabetized linked list
- Create a current node to help us traverse linked list set equal to head node of copied linked list
- While **Check List in Order by Stat (Least to Greatest)** is equal to false
    - If the current node is not equal to null and the get the next node of current node is not equal to null
        - Calls two integers which hold the **Which Stat to Get** stat of current node and node next to current
        - If stat of current node is more than stat of node next to current node

                - **Append** the current node in coped linked list // creates a copy of current node and put it at the end
                - **Delete** the first instance of the current node in copied linked list
            - Set current node equal to the node next to current node
        - If the node next to current node is equal to null // restart the traversal from the begging of the list to check if in order
            - Set the current node equal to head node of copied linked list

LinkedList Find Leaders

Parameters: Linked Lists of the sorted stats made in main, String that represents desire stat to be checked

Return: Mini linked list of leaders of that stat from parameters

- Print string of the stat from parameter
- Create mini linked list to store leaders of stat from parameters
- Create current node for traversal using head node of sorted stat
- Create integer which counts number of nodes
- While current node not equal to null
    - If the count of nodes is equal to 3
        - Break from the loop
    - Append current node to mini linked list for containing leaders for stat
    - Increment the count of nodes
- Return the mini linked list

LinkedList Output Leaders

Parameters: Mini linked list of stat, String that represents desire stat to be checked

Return:

- Create current node for traversal set equal to head node of mini linked list
- Create integer which holds amount of 1st leader ties
- While current node is not equal to null
    - If current node's player's stat is equal to head node's stat
        - Increment integer holding the amount of 1st leader ties
- Print out value of head node of stat **Which Stat to Get**
- While current node not equal to 0
    - If value of head node a.k.a the value of 1st leader stat
        - Print out the node's player name
- If integer holding 1st leader ties is equal to 3
    - Return to get out of function cause leaders are maxed out at 3 players
- Else
    - Traverse linked list by the number of times there was a 1st leader tie to put traversal in position of the 2nd leader
    - Store the stat of 2nd leader
    - Create integer which holds amount of 2nd leader ties

- While current node is not equal to null
    - If current node's player's stat is equal to 2nd leader stat
        - Increment integer holding the amount of 2nd leader ties
- Print out value of 2nd leader node player stat **Which Stat to Get**
- While current node not equal to null
    - If value of 2nd leader node player stat is equal to current node's stat
        - Print out player's name
- If integer hold amount of 2nd leader ties equal to 2
    - Return to get out of function cause leaders are maxed out at 3 players
- Else
    - Print out the 3rd node in linked list with **Which Stat to Get** because there cannot be ties for 3rd to have 3 leaders

Node Class

- Call a *Player object* which will contain stats/data of a player.
- Calls a *Node next* which will point to the next node in the list or end the list with null
- Create a **Node Constructor**
    - **Parameter:** Player object
    - *Player object* equal to *Player object parameter*
    - *Node next* equal to *null*

Player Class

Parse Player Stats

Parameter: Nothing // required data is contained within member variables a.k.a the playerStats string

Return: Nothing

- For 0 : length of player stats line
    - If character at index is equal to H
        - Integer holding player hit stat is incremented by 1
    - If character is equal to O
        - Integer holding player out stat is incremented by 1
    - If character at index is equal to K
        - Integer holding player strike out is incremented by 1
    - If character at index is equal to W
        - Integer holding player walk stat is incremented by 1
    - If character at index is equal to P
        - Integer holding player hit by pitch stat is incremented by 1
    - If character at index is equal to S
        - Integer holding player sacrifice stat is incremented by 1

// characters that do not match the above are ignored

Calculating Batting Average // this function calculates the batting average when it is necessary preventing stale data

Parameter: Nothing // required data is contained within member variables a.k.a the playerStats string

Return: double value holding the batting average

- Create integer holding the at bat value and set it equal to sum of the strike out, out, and hit stats
- If at bat value is equal to 0
    - Return 0 // prevents divide by zero scenario
- Create double batting average value holding batting average and set equal to quotient of the player hit stat divided by the at bat value

Calculating On Base Percentage

Parameter: Nothing // required data is contained within member variables a.k.a the playerStats string

Return: double value holding the on base percentage

- Create an integer holding the numerator of the on base percentage formula set equal to sum of hit, walk, and hit by pitch stats
- Create an integer holding the plate appearances (also known as the denominator of the formula) value set equal to hit, out, strike out, walk, hit by pitch, and sacrifice stat
- If numerator or denominator is equal to zero
    - Return 0
- Create double value set equal to the numerator of the on base percentage formula and divide by the casted double of the plate appearance

Adjust Decimal Point

Parameters: Double value representing calculated values

Return: String of double containing augmented floating point value

- Create and return string set equal to augmented 3 decimal place value(" %.3f", double value from parameters)

Which Stat to Get

Parameters: String which represents desired stat that the player class should get/return

Return: Nothing

- If the string is equal to H
    - Get player hit stats
    - Else if string is equal to O
        - Get player out stats
    - Else if string is equal to strikeout
        - Get player strikeout stats
    - Else if string is equal to W

        - Get player walk stats
    - Else if string is equal to P
        - Get player hit by pitch stats
    - Else if string is equal to S
        - Get player sacrifice stats

To String Method

Parameters: Nothing

Return: string which outputs player stats

- Call a string which contain all the necessary stats to display for each player

Test Cases

1) Test the linked list sort method if the players have similar names except for one character difference at the last index of their names.
2) Test the linked list check for multiple entries and have a test case where it's a bunch of entries for one player and see if all the stats are combined after processing of the file
3) Check the program if the linked list can handle leaders if there is only one player in the linked list
4) Check the sorting of stats when the leader of the stat is based on the smaller the number the better
5) Check the comparison/sort functions which dealing the different type of values such as doubles and integers, make sure it can handle both
6) Check the adaptability of the functions since I am using the same functions for different stats and make sure the stats do not conflict with a sort function for example
7) Check the processing of an individual 1st, 2nd and 3rd leader
8) Check the processing of sorting/processing when player might have missing stats
9) Check the linked list processing when given a large data set 30 or more player stat entries
10) Check the processing of leaders when there are more than 3 ties for a stat. Make sure that only 3 leaders are printed