**Linked Lists: Code Reflection and Pseudocode**

Justin Starr

Department of STEM

CS 300 – DSA: Analysis and Design

Professor Ricardo Scarello

May 26, 2024

**Linked Lists: Code Reflection and Pseudocode**

**Code Reflection**

The purpose of this code was to both manually enter bids or load a list of bids from a CSV file that would be appended to a linked list of bids that contained data for each bid (node in the linked list). The append function is used for both entering a bid and loading bids. The user is able to see a list of all bids in the linked list or search for a specific bid. The PrintList function displays all bids in the linked list. The Search function searches for a bid based on the passed in bidId. If the bid is found, that bid is returned and displayed to the user. If the bid is not found, a new empty bid is created and returned. The program then outputs to the user that the bid was not found. The Remove function removes a bid based on the passed-in bidId if the bid is found. If no bid is found, no bids are removed from the linked list. In either case, no output is displayed in the console. Lastly, the user can select to exit the application.

Implementing the various functions demonstrates an understanding of linked lists, how insertion and deletion operations work, and how the computational time to execute these functions occurs rapidly (also a main purpose of the assignment), which makes linked lists a suitable data structure for a variety of instances when adding and deleting items in a list. Having a structure of nodes that points to the next and previous nodes eliminates the costly computational time that other data structures may require such as arrays and vectors, when needing to perform these types of operations frequently because of shifting.

I faced a number of challenges while working on the assignment. Similar to last week, most of my work was done by following the comments that were provided and then determining what else might be needed that was not explicitly said in the comments to ensure the code worked as intended. The comments in the remove function were not as easy to follow as other functions. After spending time researching and working through how the remove operation needed to function, I was able to successfully implement the function. Another area, where I encountered a problem, was in the Search function. At first, I did not have an if statement to check if the head was the null pointer. Therefore, if I tried executing the search function when the list was empty, the program exited unintentionally. Once I realized what was happening, I was able to adjust and make sure this did not happen anymore. Having a sufficient amount of time to debug was critical for this assignment.

**Pseudocode**

The following is the pseudocode for the main function and the six functions that required fixing:

LinkedList::LinkedList(), void LinkedList::Append(Bid bid), void LinkedList:Prepend(Bid bid), void LinkedList::PrintList(), void LinkedList::Remove(string bidId), and Bid LinkedList::Search(string bidId).

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

DEFINE csvPath (type string) and bidKey (type string)

SWITCH (argc) {

case 2:

STORE csvPath = argv[1]

STORE bidKey = “98109”

BREAK

END case 2

case 3:

STORE csvpath = argv[1];

STORE bidKey = argv[2]

BREAK

END case 3

default:

STORE csvPath with the name of the csv file

STORE bidKey = “98109”

END DEFAULT

DEFINE ticks (type clock\_t)

DEFINE bidList (type LinkedList)

DEFINE bid (type Bid)

DEFINE choice (type int) and INITIALIZE to zero

WHILE LOOP choice not equal to 9 {

DISPLAY “MENU:” and ENDLINE

DISPLAY “1. Enter a Bid” and ENDLINE

DISPLAY “2. Load Bids” and ENDLINE

DISPLAY “3. Display all Bids” and ENDLINE

DISPLAY “4. Find Bid” and ENDLINE

DISPLAY “5. Remove Bid” and ENDLINE

DISPLAY “ Enter Choice: “ and ENDLINE

INPUT choice

SWITCH (choice) {

case 1:

STORE bid = CALL getBid() function

STORE bid to bidList with Append(Bid)

CALL displayBid(bid) to DISPLAY bid

BREAK

END case 1

case 2:

STORE ticks = CALL clock() // current clock ticks

CALL loadBids(csvPath, &bidList)

DISPLAY size of bidlist and “bids read”

STORE ticks = CALL clock() – ticks // current clock ticks

minus starting clock ticks

DISPLAY “time:”, ticks, “ clock ticks”

DISPLAY “time:”, ticks \* 1.0 / CLOCKS\_PER\_SEC, “ seconds”

BREAK

END case 2

case 3:

CALL PrintList for bidList

BREAK

END case 3

case 4:

STORE ticks = CALL clock() // current clock ticks

STORE bid = CALL Search(bidKey) in bidList

STORE ticks = CALL clock() – ticks // current clock ticks minus

starting clock ticks

IF the returned bid is not empty {

CALL displayBid(bid)

} END IF

ELSE {

DISPLAY “Bid Id”, bidKey, “not found”

} END ELSE

DISPLAY “time:”, ticks, “ clock ticks”

DISPLAY “time:”, ticks \* 1.0 / CLOCKS\_PER\_SEC, “seconds”

BREAK

END case 4

case 5:

CALL Remove(bidKey) on bidList

BREAK

END case 5

} END SWITCH

} END WHILE LOOP

DISPLAY “Good bye.”

RETURN 0

} END main function

**LinkedList::LinkedList()** function {

STORE head = nullptr

STORE tail = nullptr

} END function

**Void LinkedList::Append(Bid bid)** function {

CREATE new node temp

STORE temps data as the passed in bid

POINT temp nodes next node to nullptr

IF there is nothing at the head {

ASSIGN head node becomes temp node

ASSIGN tail node becomes temp node

} END IF

ELSE {

POINT tail node to temp node

ASSIGN tail node as new node

} END ELSE

INCRIMENT size

} END function

**Void LinkedList::Prepend(Bid bid)** function {

DEFINE temp (type node) = new Node

ASSIGN temp node with passed in bid data

IF head is the nullptr {

ASSIGN head as temp node

ASSIGN tail as temp node

POINT heads next pointer to nullptr

} END IF

ELSE {

POINT temp node to the current head node as its next node

ASSIGN head node as the temp node

} END ELSE

INCREMENT size

} END function

**Void LinkedList::PrintList()** function {

ASSIGN current node = head node

LOOP WHILE current node not equal to nullptr {

DISPLAY current node’s bidId, title, amount, and fund

ASSIGN current node to current’s next node

} END WHILE LOOP

} END FUNCTION

**Void LinkedList::Remove(string bidId)** function {

// special case if the matching node is the head node

IF the passed in bidId is equal to the head node’s bidId {

DEFINE temp (type node) and INITIALIZE with head node

ASSIGN head by pointing to head node’s next node

DELETE temp node

DECREMENT size

RETURN

} END IF

// Second case is if the node is after the head node including the tail node

ASSIGN current (type node) to head nodes next node

ASSIGN prev (type node) to head node

LOOP WHILE current node is not nullptr {

IF current node’s bidId is equal to passed in bidId {

// found the node to remove

BREAK

} END IF

ASSIGN prev node to current node

ASSIGN current node to current’s next node

} END WHILE

IF current node equals nullptr {

// a match was never found

RETURN

} END IF

ELSE a match is found {

IF current node is the tail node {

ASSIGN tail node as the prev node

} END IF

ASSIGN previous’s next node as the current’s next node

DELETE the current node

DECREMENT size

} END ELSE

} END function

**Bid LinkedList::Search(string bidId)** function {

ASSIGN current (type node) as the head ndoe

IF the head node is not nullptr {

IF the passed in bidId is the same as the head node’s bidId {

RETURN head node’s bid

} END IF

WHILE LOOP current node is not nullptr {

IF the passed in bidId is the same as the current node’s bidId {

RETURN current node’s bid

} END IF

ELSE {

ASSIGN current node to current node’s next node

} END ELSE

} END WHILE LOOP

} END IF

// the following lines are if a match is never found

IF current node is nullptr {

DEFINE newEmptyBid (type Bid)

RETURN newEmptyBid

} END IF

} END function