**3-2 Assignment: Linked Lists**

Cristiano Miranda

Southern New Hampshire University

CS-300: Analysis and Design

Professor Muthukudage

21 September 2025

**3-2 Assignment: Linked Lists**

Code Reflection:

            The main goal of this program was to load bid information from a CSV file, allow the user to enter new bids, display the bids, remove a specific bid, and search for a specific bid. The purpose of creating this program was to implement the singly linked list data structure in a program to learn how to navigate, alter, and display its data nodes. Additionally, the program required that we implement these features using multiple methods, such as adding both prepend and append functions for adding data. In doing this, we practiced the most critical features of utilizing a singly linked list. We had the chance to properly implement this data structure in situations where it was most effective. In terms of implementing the singly linked list, a class was utilized to encapsulate all the methods and attributes associated with a singly linked list, making it easier to use in the program. Using a class encapsulated all the logic and data needed to add, remove, search, and navigate the list. Within the class, two methods were added to allow for the addition of data nodes to the singly linked list. One being append, which adds a new data node to the end of the list by pointing the current tail to the new node and then setting the tail to said node, and the second being prepend, which adds the new node to the front of the singular linked list by having the new node point to the current head node and then setting the head of the list to the new node. For printing and searching, both utilize a while loop that iterates through the entire list until a null pointer is reached, but in the case of searching, the method will exit early if the appropriate data point is found. Finally, for removing elements, a singular method is used. Within the method, a base case is set before entering a while loop, in which the head of the list is confirmed to be populated with data and is checked to see if the associated bid ID is housed within it. If this is the case, the head of the list is moved to the next node, and the previous head is removed from memory. However, if this is not the case, a while executes until a null pointer is reached, and within the while loop, the next node after the current node is confirmed to be populated and checked if the associated bid is housed within it. If so, the node after the next node is stored in a variable, and the next node is freed from memory, and the current node's next pointer is then set to the temporary variable, which removes the desired node from the list. By far, implementing the remove method was the biggest challenge I faced, as it was a complex process that, if done incorrectly, could lead to a plethora of errors and easy-to-overlook problems, with one being that if the tail node of the list was removed, then the tail needed to point to the new last element of the list. To help identify all possible issues that I could overlook or encounter while implementing this method, I thoroughly consider the problem first and write the pseudocode before attempting to implement a solution. By doing this, I gave myself a better chance of avoiding errors or unforeseen issues, as thinking through the process helped me implement the necessary steps to prevent problems. For instance, I realized that I needed first to check if a pointer was not null before attempting to access data from it. In the end, my decision to take the time to think through the process and understand what I was trying to accomplish helped me account for potential errors or issues that would have caused the program to produce an error.

Pseudocode:

* CLASS LinkedList
  + INIT PRIVATE OBJECT Node ()
    - INIT bid
    - INIT next: POINTER TO Node
    - CONSTRUCTOR () // no passed in value
      * START
      * SET next TO null
      * END
    - CONSTRUCTOR (aBid) // with a passed in value
      * START
      * SET bid TO aBid
      * SET next TO null
      * END
  + END CONSTRUCTOR
  + INIT PRIVATE head
  + INIT PRIVATE tail
  + INIT PRIVATE size = 0
  + PUBLIC CONSTRUCTOR LinkedList ()
    - START
    - POINT head TO null
    - POINT tail TO null
    - END
  + END CONSTRUCTOR
  + PUBLIC DESTRUCTOR ~LinkedList ()
    - START
    - INIT current AND POINT TO head
    - INIT temp
    - WHILE current IS NOT POITING TO null
      * POINT temp TO current
      * POINT current TO current->next
      * DELETE temp
    - ENDWHILE
    - END
  + END DESTRUCTOR
  + PUBLIC METHOD Append (bid)
    - START
    - INIT new bid OBJECT using passed in bid details
    - INIT newNode AND POINT TO new bid OBJECT
    - IF (head POINTS TO null) THEN
      * POINT head TO newNode
      * POINT tail TO newNode
    - ELSE
      * POINT tail->next TO newNode
      * POINT tail TO newNode
    - ENDIF
    - INCREMENT size
    - END
  + END METHOD
  + PUBLIC METHOD Prepend (bid)
    - START
    - INIT new bid OBJECT using passed in bid details
    - INIT newNode AND POINT TO new bid OBJECT
    - IF (head POINTS TO null) THEN
      * POINT head TO newNode
      * POINT tail TO newNode
    - ELSE
      * POINT newNode->next TO head
      * POINT head TO newNode
    - ENDIF
    - INCREMENT size
    - END
  + END METHOD
  + PUBLIC METHOD PrintList ()
    - START
    - INIT currNode AND POINT TO head
    - WHILE currNode IS NOT POITING TO null
      * INIT currBid AND SET EQUAL TO currNode->bid
      * DISPLAY currBid.bidId + “: ” + currBid.title + “ | ” + currBid.amount + “ | “ + currBid.fund
      * POINT currNode TO currNode->next
    - ENDWHILE
    - END
  + END METHOD
  + PUBLIC METHOD Remove (bidId)
    - START
    - INIT currNode AND POINT TO head
    - IF currNode IS NOT POINTING TO null AND currNode->bid.bidId IS EQUAL TO bidId THEN
      * INIT sucNode AND POINT TO head->next
      * DELETE currNode
      * POINT head TO sucNode
      * IF sucNode POINT TO null THEN
        + POINT tail TO null
      * END IF
      * DECREMENT size
      * RETURN
    - ENDIF
    - WHILE currNode IS NOT POINTING TO null
      * IF currNode->next IS NOT POITING TO null AND currNode->next.bid.bidId IS EQUAL TO bidId THEN
        + INIT sucNode AND POINT TO currNode->next->next
        + DELETE currNode->next
        + POINT currNode->next TO sucNode
        + IF sucNode POINT TO null THEN

POINT tail TO currNode

* + - * + END IF
        + DECREMENT size
        + RETURN
      * ENDIF
      * POINT currNode TO currNode->next
    - ENDWHILE
    - END
  + END METHOD
  + PUBLIC METHOD Search (bidId)
    - START
    - INIT currNode AND POINT TO head
    - IF currNode IS NOT POINTING TO null AND currNode->bid.bidId IS EQUAL TO bidId THEN
      * RETURN currNode->bid
    - ENDIF
    - WHILE currNode IS NOT POINTING TO NULL
      * IF currNode->bid.bidId IS EQUAL TO bidId THEN
        + Return currNode->bid
      * ELSE
        + POINT currNode TO currNode->next
      * ENDIF
    - ENDWHILE
    - INIT emptyBid AND SET TO new bid OBJECT // creates an empty bid object
    - RETURN emptyBid
    - END
  + END METHOD
  + PUBLIC METHOD Size ()
    - START
    - RETURN size
    - END
  + END METHOD
* FUNCTION displayBid (bid)
  + START
  + DISPLAY bid.bidId + “: ” + bid.title + “ | ” + bid.amount + “ | “ + bid.fund
  + RETURN
  + END
* END FUNCTION
* FUNCTION getBid ()
  + START
  + INIT bid
  + DISPLAY “Enter Id: “
  + GET bid.bidId
  + DISPLAY “Enter title: “
  + GET bid.title
  + DISPLAY “Enter fund: “
  + GET bid.fund
  + DISPLAY “Enter amount: “
  + SET bid.amount
  + RETURN bid
  + END
* END FUNCTION
* FUNCTION loadBids (csvPath, LinkedList)
  + START
  + DISPLAY “Loading CSV file “
  + INIT file

// read file associated with file path

* + READ from file at csvPath and SET file to result
  + TRY
    - FOR index from 0 to number of rows in file
      * INIT bid

// Set all associated properties via their respective row and column location in the CSV file

* + - * SET bid.id = file[index][1]
      * SET bid.title = file[index][0]
      * SET bid.fund = file[index][8]
      * SET bid.amount = file[index][4]
      * APPEND bid to LinkedList
    - ENDFOR
  + CATCH (Error)
    - DISPLAY Error message
  + ENDTRY
  + END
* FUNCTION strToDouble (str, ch)
  + START
  + ITERATE through string and ERASE all instances of ch character variable and SET str to result
  + RETURN str
  + END
* END FUNCTION
* FUNCTION main (argc, argv)
  + START
  + INIT csvPath AND bidKey
  + IF argc IS EQUAL TO 2 THEN
    - SET csvPath TO argv[1]
    - SET bidKey TO “98109”
  + ELSE IF argc IS EQUAL TO 3 THEN
    - SET csvPath TO argv[1]
    - SET bidKey TO argv[2]
  + ELSE
    - SET csvPath TO "eBid\_Monthly\_Sales.csv"
    - SET bidKey TO “98109”
  + ENDIF
  + INIT ticks
  + INIT bidList
  + INIT bid
  + INIT choice AND SET TO 0
  + WHILE choice IS NOT EQUAL TO 9
    - DISPLAY “MENU:”
    - DISPLAY “ 1. Enter a Bid”
    - DISPLAY “ 2. Load Bids”
    - DISPLAY “ 3. Display All Bids”
    - DISPLAY “ 4. Find Bid”
    - DISPLAY “ 5. Remove Bid”
    - DISPLAY “ 9. Exit”
    - DISPLAY “ Enter choice: ”
    - GET choice
    - IF choice IS EQUAL TO 1 THEN
      * CALL getBid () AND SET bid TO result
      * APPEND bid TO bidList
      * CALL displayBid(bid)
      * BREAK
    - ELSEIF choice IS EQUAL TO 2 THEN
      * CALL clock AND SET ticks TO result
      * CALL loadBids (csvpath, bidList)
      * SET ticks = clock() - ticks
      * DISPLAY “time: ” + ticks + “ clock ticks”

// convert the clock time to seconds using a predefined constant variable

* + - * DISPLAY “time: ” + ticks \* 1.0 / constant variable CLOCKS\_PER\_TICK + “ seconds”
      * BREAK
    - ELSE IF choice IS EQUAL TO 3 THEN
      * CALL bidList.PrintList ()
      * BREAK
    - ELSE IF choice IS EQUAL TO 4 THEN
      * CALL clock AND SET ticks TO result
      * CALL bidList.Search (bidKey) AND SET bid TO result
      * SET ticks = clock() – ticks
      * IF bid.bidId IS NOT EMPTYTHEN
        + CALL displayBid (bid)
      * ELSE
        + DISPLAY “Bid id ” + bidKey + “ not found.”
      * ENDIF
      * DISPLAY “time: ” + ticks + “ clock ticks”

// convert the clock time to seconds using a predefined constant variable

* + - * DISPLAY “time: ” + ticks \* 1.0 / constant variable CLOCKS\_PER\_TICK + “ seconds”
      * BREAK
    - ELSE IF choice IS EQUAL TO 5 THEN
      * CALL bidList.Remove (bidkey)
      * BREAK
    - ELSE IF choice IS EQUAL TO 9 THEN
      * BREAK
    - ELSE
      * DISPLAY “Invalid Input”
      * SET choice TO 0
      * CLEAR user input
      * BREAK
    - ENDIF
  + ENDWHILE
  + DISPLAY “Good bye.”
  + RETURN
  + END
* END FUNCTION