# Thomas Martin

# CS-300-H2990 DSA: Analysis and Design 23EW2

# 3-2 Assignment: Linked Lists- Code Reflection and Pseudocode

# Southern New Hampshire University

**November 18, 2023**

**Code Reflection**

The LinkedList object sets the head and tail pointers to nullptr, indicating an empty list. It also initializes the size variable to 0.

One common issue is ensuring that you correctly manage the access specifiers private and public and handle data encapsulation.

In the LinkedList Append (Bid), the bid is passed. The function then adds the past Bid as a new Node. The new Node is placed at the end of the list and updates the head and tail.

The code maintains consistency by using similar naming conventions for appending at the end.

The Prepend function prepends a new bid to the start of the list. It creates a new node with the provided bid and adds it to the beginning of the list. If the list is not empty, the new node is attached to the current head, and the head is updated to the new node. The new node becomes the head and tail if the list is empty.

Testing the code thoroughly with various scenarios is important. This includes testing the prepend operation with an empty list, adding bids to an already populated list, and verifying that the list is correctly updated.

The function LinkedList PrintList defines a bid entry to represent a bid entry, containing a string for the bidder's name and an integer for the bid amount.

Developing this code is a standard practice in C++ and involves working with vectors and structs.

The LinkedList Remove function starts at the head and searches for the passed String. Upon finding the string the node containing it is freed from memory. The head and tail are updated accordingly if either is the node containing a string.

Developing this code involves extending the existing functionality to include bid IDs and implementing bid removal.

It's important to test the search logic with various scenarios, including searching for bids that exist and don't exist.

**Pseudocode**

Default Constructor:

LinkedList::LinkedList() {

SET head and tail to null pointer

END

}

LinkedList::Append(Bid bid) {

INITIALIZE Node\* node to a new node containing bid

IF head is equal to null pointer

SET head and tail to node

ELSE

POINT tail to node

SET tail to node

INCREMENT size of the list

END

}

LinkedList::Prepend(Bid bid) {

INITIALIZE Node\* node to a new node containing bid

IF head is equal to null pointer

SET head and tail to node

ELSE

POINT node to head

SET head to node

INCREMENT size of the list

END

}

LinkedList::PrintList() {

INITIALIZE Node\* current to head

While current node is not null pointer

PRINT the current information

SET current to current.next

END

}

LinkedList::Remove(string bidId) {

INITIALIZE Node\* current to head

If current is not null pointer and bidId is equal to current's bid ID

SET head point to current.next

DELETE current

DECREMENT the list size

RETURN

END

WHILE current.next is not null pointer

IF bid ID of current.next is equal to bidId

INITIALIZE Node\* temp to current.next

SET current.next point to current.next.next

DELETE temp

DECREMENT the list size

RETURN

END

SET current to current.next

RETURN

END

}

LinkedList::Search(string bidId) {

INITIALIZE Node\* current to head

WHILE current is not null pointer

IF current's bid ID is equal to bidId

RETURN current's bid

END

SET current to current.next

RETURN empty bid

END

}