**Code Reflection**

The purpose of this project was to practice working with **singly linked lists** by implementing a custom data structure to store and manage auction bids from a CSV file. The linked list was chosen because it allows efficient insertion and deletion without needing to shift data, which is useful for managing dynamic sets of records. To complete the project, I implemented methods for appending bids to the end, prepending to the beginning, searching for a bid by ID, removing a bid by ID, and printing the entire list. Each method followed the linked list traversal pattern, using pointers to step through the nodes until the correct location was found.

One challenge was handling **edge cases**, such as removing the head node, searching for a non-existent bid, or printing when the list is empty. These were addressed by checking for null pointers before accessing node data. Another challenge was ensuring that nodes were properly deleted to avoid memory leaks, which I handled by explicitly calling delete when removing bids. Through this assignment, I reinforced the importance of careful pointer management and writing modular code so that each method serves a single, well-defined purpose.

Class LinkedList:

Node structure:

Bid data

Node\* next

Variables:

Node\* head

Node\* tail

int size

Constructor:

Set head = null

Set tail = null

Set size = 0

Method Append(bid):

Create newNode with bid

If head is null:

head = newNode

tail = newNode

Else:

tail->next = newNode

tail = newNode

size++

Method Prepend(bid):

Create newNode with bid

newNode->next = head

head = newNode

If tail is null:

tail = newNode

size++

Method PrintList():

node = head

While node != null:

Print node.bidId, node.title, node.amount, node.fund

node = node->next

Method Search(bidId):

node = head

While node != null:

If node.bidId == bidId:

Return node.bid

node = node->next

Return empty bid

Method Remove(bidId):

If head == null: return

If head.bidId == bidId:

temp = head

head = head->next

Delete temp

size--

Return

prev = head

node = head->next

While node != null:

If node.bidId == bidId:

prev->next = node->next

If node == tail:

tail = prev

Delete node

size--

Return

prev = node

node = node->next