**Hash Table Assignment Reflection**

This hash table implementation stores bid data using chaining to handle collisions. The main goal was to efficiently insert, search, and remove bids by hashing their IDs into table indices. I used a simple modulo-based hash function and linked lists for handling collisions—each bucket points to a chain of nodes if multiple bids hash to the same index. The trickiest part was keeping tracks of nodes during insertion and deletion, especially when removing nodes from the middle of a chain. I overcame these by carefully reading the pseudocode instructions and reading text book to make sure the logic is correct.

**Pseudocode for Insert() :**

// create the key for the given bid

// retrieve node using key

// if no entry found for the key

// assign this node to the key position

// else if node is not used

// assing old node key to UNIT\_MAX, set to key, set old node to bid and old node next to null pointer

// else find the next open node

// add new newNode to end

**Pseudocode for PrintAll() :**

// for node begin to end iterate

// if key not equal to UINT\_MAx

// output key, bidID, title, amount and fund

// node is equal to next iter

// while node not equal to nullptr

// output key, bidID, title, amount and fund

// node is equal to next node

**Pseudocode for Remove (string bidId) :**

// set key equal to hash atoi bidID cstring

// erase node begin and key

**Pseudocode for Search (string bidId) :**

// create the key for the given bid

// if entry found for the key

//return node bid

// if no entry found for the key

// return bid

// while node not equal to nullptr

// if the current node matches, return it

//node is equal to next node