**Reflection**

The purpose of this program was to implement a hash table with chaining to manage a collection of bids for an auction system. The program was designed to allow for the efficient insertion, removal, searching, and printing of bids using a hash-based indexing system. Each index in the table holds a linked list to handle collisions. A modulo-based function is used to map bids to indices. Dynamically allocated nodes are created to construct linked lists within each bucket for chaining. The program is designed to load bids from a CSV file and allows custom file entry along with validation for effective error handling. The challenges encountered in this project primarily involved table and list manipulation. Ensuring the program identifies the correct data for either printing or deleting, as well as guaranteeing that insertion is collision-free, was essential. The insertion, removal, and search operations of the table have an average efficiency of O(1) due to effective hashing, while printing has an efficiency of O(n+k) because it requires cycling through all the table's buckets (n) and the subsequent bids found in those buckets (k).

**Pseudocode**

function HashTable():

set tableSize to DEFAULT\_SIZE (179)

resize the nodes vector to tableSize

function ~HashTable():

for each index i from 0 to tableSize:

set current to nodes[i].next

while current is not null:

set temp to current

move current to next node

delete temp node

clear the nodes vector

function hash(key):

if tableSize is 0:

return 0

return key modulo tableSize

function Insert(bid):

compute index = hash(bid.bidId as integer)

if nodes[index] is unused (key == UINT\_MAX):

assign bid to nodes[index] with key

else:

set cursor to nodes[index]

while cursor.next is not null:

move cursor to next node

cursor.next = new Node with bid and key

function PrintAll():

for each index i from 0 to tableSize:

set cursor to nodes[i]

if cursor is valid (key != UINT\_MAX):

while cursor is not null:

print cursor.key, bidId, amount, fund

move cursor to next node

function Remove(bidId):

compute index = hash(bidId as integer)

set cursor to nodes[index]

set prev to null

while cursor is not null:

if cursor's bidId matches bidId:

if prev is null (removing head node):

if cursor has a next node:

copy next node data into cursor

delete next node

else:

reset cursor to unused state (key = UINT\_MAX, empty bid)

else:

prev.next = cursor.next

delete cursor

print success message

return

move prev and cursor to next nodes

print not found message

function Search(bidId):

compute index = hash(bidId as integer)

set cursor to nodes[index]

while cursor is not null:

if cursor's bidId matches bidId:

return cursor's bid

move cursor to next node

return empty Bid object