4-2 Assignment: Hash Tables

Fred Wahab

CS-300

**Code Reflection**

This week’s assignment had us implementing a singly linked list to hold a collection of bids loaded form a .csv file. A hash table uses a hash function to compute an index, also called a hash code, into an array of buckets or slots, from which the desired value can be found. The purpose of hash tables is to make a search function easier for the program. A benefit to hash tables is that the elements get put into a table and a chain. This makes it easier to look for an item. Converting large keys into smaller keys by using hash functions will achieve the goal. Calling hash tables is complete after the process is complete.

Like most weeks, the biggest issue I had was deciphering the instructions within the starter code. I have a routine now where I create the project, import the start code, and repair the errors that come with the starter code so I can test as I go. Then I had to go through each fix-me and try to figure out what it wants me to do.

**Pseudocode**

**FIXME (1): Initialize the structures used to hold bids**

HashTable::HashTable() {

Resize HashTable to size of node

}

**FIXME (2): Implement logic to free storage when class is destroyed**

HashTable::~HashTable() {

Erase nodes beginning

}

**FIXME (3): Implement logic to calculate a hash value**

unsigned int HashTable::hash(int key) {

Return key tableSize

}

**FIXME (5): Implement logic to insert a bid**

void HashTable::Insert(Bid bid) {

create the key for hash(atoi(bid.bidId.c\_str()))

Retrieve node using key (nodes.at(key))

if oldNode equals null, then assign this node to the key position

else if oldNode is not found, then assign this node to the key position

else find the next open node and add node to the end

}

**FIXME (6): Implement logic to print all bids**

void HashTable::PrintAll() {

initialize variable i with 0

while i is less than the nodes’ size, call displayBid function and increment i by 1

}

**FIXME (7): Implement logic to remove a bid**

void HashTable::Remove(string bidId) {

set key equal to hash(atoi(bidId.c\_str()))

erase node begin and key

}

**FIXME (8): Implement logic to search for and return a bid**

Bid HashTable::Search(string bidId) {

set key equal to hash(atoi(bidId.c\_str()))

set node equal to &(nodes.at(key));

if node equals nullptr or node->key equals UINT\_MAX, then return bid

if node does not equal nullptr and node->key does not equal UINT\_MAX and node- >bid.bidId.compare(bidId) equals 0, then return node->bid

while node does not equal nullptr {

if node->key does not equal UINT\_MAX and node->bid.bidId.compare(bidId) equals 0 {

return node->bid;

}

return bid;

}