Module Four Assignment

By: Darrell Walker II

The purpose of this code is to create a hash table, which is a data structure designed to efficiently store and retrieve data using a technique called hashing. In this specific case, the hash table is used to manage bids for property auctions. The hash table uses a method called chaining to handle collisions, which occur when two bids hash to the same index in the table. Chaining involves creating a linked list at each index to store multiple bids.

The techniques implemented to solve the problem include hashing to calculate an index from the bid ID, and chaining to handle collisions by linking nodes in a list at each table index. The code also dynamically allocates and frees memory as needed, ensuring efficient use of resources. Challenges encountered included correctly handling collisions, ensuring efficient search and retrieval, and managing memory to avoid leaks. These were overcome by implementing robust linked list operations for insertion, deletion, and traversal, and by including detailed comments and modular functions to keep the code organized and maintainable.

Task 2: Initialize the structures used to hold bids.

Function HashTable()

Resize the nodes vector to the default table size

End Function

Function HashTable(size)

Set the table size to the specified size

Resize the nodes vector to the new table size

End Function

Task 3: Implement code to free storage when a class is destroyed.

Function ~HashTable()

Clear all elements in the nodes vector

End Function

Task 4: Implement code to calculate a hash value using the bid ID as the source for calculating the key.

Function hash(key)

Return key modulo table size

End Function

Task 5: Implement code to insert a bid. Be sure to check for key collisions and use the chaining technique with a linked list to store the additional bids.

Function Insert(bid)

key = hash(convert bid ID to integer)

node = nodes[key]

If node's key is not set (equals UINT\_MAX)

Set node's key to the calculated key

Set node's bid to the given bid

Set node's next to null

Else

Create a new node with the bid and key

Set the new node's next to the current node's next

Set the current node's next to the new node

End Function

Task 6: Implement code to print all bids.

Function PrintAll()

For each node in the nodes vector

If the node's key is set (not UINT\_MAX)

Print the node's bid details

current = node's next

While current is not null

Print current node's bid details

Move to the next node

End Function

Task 7: Implement code to remove a bid.

Function Remove(bidId)

key = hash(convert bid ID to integer)

node = nodes[key]

current = node's next

If node's bid ID matches the given bid ID

Replace node with its next node

Else

While current is not null and current's bid ID does not match

Move to the next node

If current is not null (match found)

Set previous node's next to current's next

Delete the current node

End Function

Task 8: Implement code to search for and return a bid.

Function Search(bidId)

key = hash(convert bid ID to integer)

node = nodes[key]

While node is not null

If node's bid ID matches the given bid ID

Return the node's bid

Move to the next node

Return an empty bid (not found)

End Function