### **Dylan Lovin**

SNHU CS-300

2/2/2025

### **Module 4: Hash Table**

This module focuses on implementing a hash table, structured as follows:

#### **Class: HashTable**

The HashTable class manages hashed data storage and includes:

* Struct: Node
  + Contains a Bid structure
  + An integer key
  + A pointer to the next node for handling collisions
* Methods:
  + HashTable() – Default constructor that sets the size of the node vector
  + hash(key) – Computes a hash using the modulo method, where the bid ID acts as the key
  + Insert(Bid) – Adds a new Bid to the hash table. If a collision occurs, it appends the new node at the end of the chain
  + PrintAll() – Iterates through the table and displays four key values from each Bid
  + Remove(String) – Locates and removes a specified node from the hash table while maintaining the chain integrity
  + Search(String) – Searches for a node by its key and returns it if found
  + Size() – Retrieves the total number of elements in the hash table

#### **Additional Functions:**

* strToDouble – Converts CSV data into a usable numeric format
* Bid – A struct that holds bid-related data, later used for sorting
* displayBid – Outputs bid values from the vector to the console
* loadBids – Reads bid data from a CSV file, either from an argument-specified path or a default location, then inserts it into the hash table

#### **Main Function:**

The main function acts as the core of the program, offering a menu-driven interface that allows users to:

* Enter new bids
* Load bid data from a CSV file
* Display stored bids
* Delete entries
* Exit the application

Additionally, time.h is used to measure the execution time of various operations.

### **Personal Experience:**

Overall, the code was straightforward, especially since the parsing functionality was pre-provided. However, I encountered an issue when adding nodes to the hash table. Despite correct syntax, my IDE flagged errors until I restarted my virtual machine. The cause remains unclear.

**Pseudocode:**

### **Main Function()**

* Read command-line arguments.
* Store the argument as the file path for the CSV.
* If no argument is provided, use the default CSV file path.
* Enter a loop until the user selects option '9'.
  + Display the menu.
  + Receive and store user input as choice.
  + Validate the input:
    - If choice is not 1, 2, 4, or 9, return an error.
  + If choice is '1':
    - Start the timer and store the start time.
    - Call loadBids() to read data from the CSV file into bidTable.
    - Display the total number of records loaded.
    - Stop the timer.
    - Display the elapsed time for reading the file.
  + If choice is '2':
    - Call PrintAll() to display the stored bids.
  + If choice is '4':
    - Start the timer and store the start time.
    - Call Search() with a specified bidKey.
    - Stop the timer.
    - Display the elapsed time for finding bidKey.
  + If choice is '4' again (remove option):
    - Call Remove() with the bidKey.
  + If choice is '9':
    - Exit the program.
    - Display "Goodbye".

### **HashTable::hash(int key)**

* Return key % tableSize.

### **HashTable::Insert(Bid bid)**

* Compute the hash index using hash(bid.bidId), store the result in tempKey.
* Check if the corresponding position in the table is empty:
  + If empty, create a new node containing Bid and tempKey, and assign it to the hash location.
  + Otherwise:
    - Create a node pointer referencing the current position in the table.
    - Create a new node with Bid and tempKey.
    - Traverse the linked list at that location until reaching the end, then append the new node.

### **HashTable::PrintAll()**

* Create a node pointer referencing the first node in the table.
* Traverse the list from the beginning:
  + If the key value at the current position is not UINT\_MAX:
    - Print bidId, title, amount, and fund to the console.

### **HashTable::Search(String bidKey)**

* Create a node pointer cursor and set it to the bucket at the hashed index.
* Traverse the linked list at that location:
  + If the bidId of the node at cursor matches bidKey:
    - Return the cursor node.
  + Otherwise, move to the next node.

### **HashTable::Remove(String bidKey)**

* Create a node pointer cursor and set it to the bucket at the hashed index.
* Create another node pointer tempNode.
* Determine if cursor is part of a chain or a single bucket:
  + If it's part of a chain:
    - Check if cursor->bidId matches bidKey:
      * If so, set tempNode to the next node.
      * Update cursor to point to tempNode.
      * Delete tempNode.
    - If not at the start of the chain, iterate through the list to find a match.
  + If it's a single bucket:
    - Reset the bucket by assigning default values to its Bid members.