Billy Hegarty

CS300 Assignment 4: Hash tables

## Code Reflection

The code “HashTable.cpp” reads in data from a .csv file and provides several data engagement choices to the user via a numerical menu. The .csv file location can be specified as a command line argument, otherwise it will default to a file named “eBid\_Monthly\_Sales\_Dec\_2016.csv" located in the same local file as LinkedList.exe. The .csv file contains rows of data. Each row corresponds to an individual bid and auction for an item, such as computer parts or office furniture. The program organizes each bid in a data structure named “Bid” and assigns attributes bidId, title, fund, and amount. These attributes are automatically read in from the .csv file using the provided “CSVparser.cpp” file by selecting menu option #1.

When the program runs, the main() function declares a HashTable object named “bidTable” that will eventually be used to store all data from the .csv file. Each node in the hash table is initialized to hold a default node construction, a default key, and a “next” pointer with a null destination.

Menu option #1, “Load Bids”, reads in one line at a time from the .csv file. After reading in each line, the bid data is inserted into a hash table using the Insert() method. This method reads in the bid ID#, translates it from a string to an integer, and calls the hash() method on that integer to obtain a hash key. The hash() method utilizes simple modulus division to calculate the remainder when dividing the bid ID# by the hash table size. In this program, the table size is set to 179. As each bid is read in, the Insert() method first checks whether or not the resultant key has already been accessed in the hash table. If it has not, the Insert() method updates the default node at that key location with the actual bid data and the correct key that was obtained from hashing the bid ID. If the node was previously accessed, then the program employs chaining to handle the collision. The bid data is inserted as a new node at the end of a linked list at that key location.

Menu option #2, “Display All Bids”, prints a header line with column titles, then prints each node in the hash table that contains non-default bids (i.e. it prints the bids that were read in from the .csv, but not blank bids that were created to populate the table when the hash table was declared).

Menu option #3 is Find Bid. The program will search for a bid with a particular bidID, either specified in as a command line argument or by the default ID value 98223. To accomplish this function, the code performs a hash of the provided bid ID, then navigates to that key location in the hash table. The Search() method then traverses down the linked list (which might only contain the head node and nothing else) until it finds a node containing a bid with a matching bid ID. It starts by checking whether the first node in the list (the head) matches the target ID. If this does not result in a match, the program continues by assigning a temporary node named “current” to point to the next node. The search continues until a match is found, or until the current pointer points to a null node. If the matching bid is found, the program prints the contents of the bid. If the bid is not found, the Search() method prints a message to alert the user and returns a blank bid.

Menu option #4 is Remove Bid. Similar to Find Bid, this method uses the particular bidID passed as a command line argument, or the default 98223 value. Find Bid navigates to the appropriate node in the hash table, then traverses the linked list there (if present). Assuming the bid exists in the hash table, there are three possibilities. First, the target bid is the only bid in the key node (no collisions occurred at this key, thus no linked list exists). In this case, the key node is populated with a default bid construct, thus “removing” the target bid. The second possibility is that the target bid is the head node of a linked list at the key node position. In this case, the contents of the second node are copied to the first node, the address of the target bid is assigned to the second node, and the second node is then deleted by deleting the address. The head node still points to what used to be the third node, so the linked list is preserved, but the target bid was removed in the process. The third case is that the target bid exists somewhere deeper in the linked list. In this case, the previous node’s “next” pointer is reassigned to point at the target bid’s “next” pointer. This effectively cuts out the target bid from the linked list, and the target bid address can be safely deleted.

I used a JetBrains IDE called CLion instead of Microsoft Visual Studio. CLion is user friendly and provides visual indications of errors, warning, and typos throughout the code. Since I had previously made housekeeping edits to the “CSVparser.cpp” and “CSVparser.h” files during the last assignment, I reused those clean files instead of using the helper files provided for this assignment.

I ran into several road blocks when trying to code this program. First, the Search() method was not returning the target bid, but I discovered that the default value provided in main() is not actually contained in the sample data. After picking a new default bid ID# in main(), the Search() method worked perfectly. The Remove() method was much trickier. First, I encountered an error code that translated to STATUS\_ACCESS\_VIOLATION. After debugging, I discovered that I was attempting to dereference a null pointer. This was occurring in the scenario where I tried to remove a bid from a linked list that only contained one item. I corrected that part of the code, but then encountered a new message, STATUS\_HEAP\_CORRUPTION. This error message took the most time to solve. In fact, I spent more time combatting this one error message than I did on the entire remainder of the program. Eventually, I discovered that my Remove() function was attempting to delete the node held at the “current” pointer, regardless of its status in the hash table. It was fine to delete a “current” pointer after rearranging a linked list, but my code resulted in deleting a “current” pointer that still pointed to the head node in the hash table. It was interesting that this error was not a result of a typo, or syntax error, or something related to the code itself, but actually it was rooted in my fundamental understanding of the hash table structure. Eventually I drew out the hash table on a blank sheet of paper, including several scenarios (default nodes, single bid nodes, and nodes containing a linked list of bids). This visual representation, together with stepping through the code in the debug menu, really helped me understand what was actually happening in the Remove() method and all other methods.

## Pseudocode

* Define a structure “Bid” with attributes that correspond to the column headers in the data .csv file.
* Declare functions that will be defined later
  + strToDouble
  + displayBid, a function to display the attributes of a Bid type
* Define the HashTable class
  + Define a structure “Node”, which will contain Bid data as well as a pointer to the next node
  + Define default constructor for Node, and populate it was a default (empty) Bid
  + Declare public members for the HashTable class
* Define default constructor for HashTable class
  + Resize to tableSize (defaulted to 179)
* Define constructor for HashTable class, given a table size
* Define default destructor for HashTable class
  + For each node in the HashTable, traverse the linked lists in each node and delete the child nodes of the linked list to free up memory
* Define hash() method
  + Return hash key by modulo against the tableSize
* Define Insert method to HashTable class
  + Given a new bid to insert into a linked list, create a hash key for that bid
  + Retrieve the node at that key
  + If that node contains default data, replace node with bid data
  + If that node contains a bid, handle the collision by inserting the bid data on a new node in a linked list:
    - Traverse the linked list until finding a null pointer
    - Have that pointer point to a node containing the new bid data, instead of null pointer.
* Define a method to print all bids in the hash table
  + Start with the first node
  + If the node is non-default, print the node bid data
  + If that bid data points to another bid (linked list within the node), traverse that linked list and print subsequent bids until reaching the tail
  + Move to the next node and repeat
* Define Remove method to HashTable class
  + Given a target bid ID number, calculate the hash key
  + Assign a pointer to the hash key node
  + While that pointer is not null:
    - If the node contains the target bid and no linked list, replace the node with a default bid
    - If the node contains the target bid in the head node, copy the next node data into the head node, set the target bid address to what was previously the second node, then delete the target bid address
    - If the node contains the target bid somewhere else in the linked list, assign the previous pointer to point to the next node, thus bypassing the target bid, then delete the target bid address.
  + If a match is not found, check the next node, and so on until reaching the end of the hash table
  + Print a message if the target bid was removed
  + Print a message if the target bid was not found
* Define Search method to HashTable class
  + Given a target bid ID number, calculate the hash key
  + Assign a pointer to the hash key node
  + While the pointer is not null
    - Compare the bid ID in the node to the target bid ID
    - If it matches, print a message a return (stop the method)
    - If it is not a match, move on to the next node in the linked list
  + If the pointer becomes null, print a message that the bid was not found
* Define a method to display bid attributes
* Define a function called “loadBids”
  + Input parameter string for the CSV file path
  + Create bid objects for each line in the CSV file
  + Insert the bids into the hash table using Insert()
* Define a function strToDouble that strips an unwanted character from a string and then converts the string to a double
* Create a main function:
  + Process command line arguments, such as the filepath of the CSV file
  + Create a hash table that will contain bids
  + Define a timer variable
  + Display a menu and get user choices
  + Perform actions based on user choice
    - Load Bids
    - Display All Bids
    - Find Bid
    - Remove Bid
    - Exit
  + Display a goodbye message after exiting