**Code Reflection week 4**

**Jaden Bryon Knutson**

-------------------------------------------------------------------------------------------------------------

**A brief explanation of the code and its purpose:**

**Description: Hash Table Bids –**

* The given program is a C++ implementation of a hash table with chaining.
* It includes header files, global definitions, and a structure to hold bid information.
* The program defines a class called `HashTable`.
* The class has methods to insert a bid, print all bids, remove a bid, and search for a bid based on its bidId.
* The program also includes additional functions to:
* display a bid,
* load bids from a CSV file into the hash table,
* and convert a string to a double.
* The main function provides a menu-driven interface to interact with the hash table, including options to:
* load bids,
* display all bids,
* find a bid,
* remove a bid,
* and exit the program.
* I have added in menu options 6 and 7 so the user can choose between the two bid files.
* Option 6 - Switches the load directory to: eBid\_Monthly\_Sales.csv
* Option 7 - Switches the load directory to: eBid\_Monthly\_Sales\_Dec\_2016.csv
* Once the new directory is switched, the user can reload by entering "1" and can now sort/display the new file.

-------------------------------------------------------------------------------------------------------------

**Brief discussion of your experience in developing it & issues encountered during exercise and approaches to solve them:**

For the most part, this week went well. I encountered some difficulties in figuring out how to output the bids correctly. Initially, I managed to complete most of the tasks, but then I had to refer to the video to figure out some errors. Towards the end, my main issue was that the "key" was being printed when it shouldn't have been. The solution turned out to be quite simple and it shouldn't have taken as much time as it did.

Currently, most of my code resembles the code shown in the video, but it did look a bit different before. I’ll include a non-working version of my code that I was previously working on as part of the documentation. The output may be a bit messy and there could be some other details that need fixing, but this acted as my tester file.

-------------------------------------------------------------------------------------------------------------

**A pseudocode description of the code that is clear and understandable and captures accurate logic to translate to the programming language:**

**Default constructor:**

* Initialize data structure used to hold bids.
* Resize nodes vector to match table size.

**Constructor for specifying size of table:**

* Set size of table to specified value.
* Resize nodes vector to match specified size.

**Destructor:**

* Free storage used by class when destroyed.
* Clear nodes vector to release storage.

**Calculate hash value of given key:**

* Calculate hash value for given key by performing modulo operation with table size.
* Return calculated hash value.

**Insert bid:**

* Create key for given bid by calculating hash value using bid's bidId.
* Retrieve node using key.
* If no entry found for key,
* create new node with bid and insert it at key position in nodes vector.
* If node is already used,
* find next available node by traversing linked list.
* Add new node at end of linked list.

**Print all bids:**

* Iterate over each node in nodes vector.
* If key of node is not equal to special value,
* print key, bidId, title, and fund of bid.
* Traverse linked list of nodes (starting from next node of current node) and print their information as well.

**Remove bid:**

* Calculate key for bidId using hash function.
* Find node at beginning of key in nodes vector.
* If first node's bidId matches given bidId,
* remove it by marking key as empty or replacing it with next node.
* If first node does not match bidId,
* search for matching node in linked list and delete it if found.

**Search for specified bidId:**

* Create key for given bidId by converting it to an integer and calculating hash value.
* Get node corresponding to key from nodes vector.
* Check if first node matches bidId. If so,
* return bid.
* If no entry found for key,
* return empty bid.
* Traverse linked list of nodes, checking if each node matches bidId. If match found,
* return bid.
* If no matching bid found,
* return empty bid.