Jack Thake
Karla Fant
CS163
11/4/20

Program 3 Design Writeup

In Program 3 two data structures are needed, a Queue and a Hash Table. Although these abstract data types aren't intertwined and don't communicate with each other. The data in these two data types will be a structure representing a single item in a Halloween treasure hunt. The programmer will dequeue the next item to find from the queue and then look up the item in the hash table, the hash table then returning that item's detail: name, location, hints etc. I expect the main challenge is finding the right balance in the hash table, finding the right hash function and the right array length. Another being I'm not very experienced with Circular Linked Lists and that will have its own set of challenges.

Denoting Success and Failure

As with all of my previous programs, success and failure will be shown by a typedef-ed enum called CS_error, this type will hold the following error or success states:

- **SUCCESS**: The operation succeeded.
- **FAILURE**: The operation failed.
- **MEM_ALLOC_FAIL**: Memory failed to be allocated.

File Handling

` In this program, all of the items in the treasure hunt are loaded from an external data file, items.dat. This file will have the information needed to create one CS_item_info structure per line. With each attribute separated by a '|' character. The format would look like the following:

Item Name|Location of Item|Hint|Rating

Along with this will be a single function to read the data file, this function will simply look for the above-named data file, load its contents, and return a filled out queue and hash table. This will be run at the start of the program before any user interaction as a way to set the test program up.

CS_item_info Structure

In this assignment as said above, the items are represented by a struct: ***CS_item_info***. This struct will also have overloaded operator equals, as well as an overloaded equality operator to aid ease of use in the Abstract Data Types. This will contain the following data members as well as their respective types:

- **Name:** *String* - The name of the item.
- **Location:** *String* - The exact location of the item.
- **Hints**: *String* - A hint on where the item is.
- **Rating**: *Integer* - A 0 to 5 rating of the item from past treasure hunters.

CS_item_queue Class

The CS_item_queue class will contain a Circular Linked List of ***CS_item_info*** structures. Its job will be to simply store the items and dequeue them back to the application programmer when prompted. To achieve this it will contain the following member functions:

- **Constructor**: Initiate the queue, set up member variables.
- **Destructor**: Deallocate all remaining dynamic memory.
- **Enqueue**: Add to the end of the queue. This will take in a ***CS_item_info*** struct and will add it to the queue, will return **SUCCESS** upon completion or **MEM_ALLOC_FAIL** if memory failed to be allocated.
- **Dequeue**: Remove from the queue, returning the removed item. Returns **SUCCESS** upon completion. Returns **FAILURE** if the queue is empty.
- **Peak**: Return the first item in the queue, making sure the application programmer cannot modify it and the queue, ie give back a copy. Returns **SUCCESS** upon completion, **FAILURE** if the queue is empty.
- **Is Empty**: Just returns a simple boolean flag denoting whether the queue is populated or not.

CS_item_table Class

The ***CS_item_table*** class will at its core to be a hash table. To resolve collisions in the hash function, chaining will be used. Chaining is a collision resolution technique where each array index in the table is itself a Linear Linked List when a collision occurs the item is just added to the list at the given index. Because of this technique, much care needs to be taken with coming up with the hash function(s) and the array length itself. This program has a more experimental feel to it, one of the points being to find

the correct balance for an efficient hash table implementation. Because of this I will implement two hash functions and experiment with different array lengths. The core functionality of a hash table is as follows:

- **Constructor**: Initialize array of lists.
- **Destructor**: Deallocate all lists.
- **Add item**: Adds a given ***CS_item_info*** structure using its name as the key, passing that key to the internal hash function. Returning **SUCCESS** on completion, **MEM_ALLOC_FAIL** if memory could not be allocated.
- **Remove item**: Remove a specific item based on its name, returns **SUCCESS** on completion **FAILURE** otherwise.
- **Remove all**: Remove all items from the table. Returns **SUCCESS** upon completion, **FAILURE** if the list is empty.
- **Retrieve item**: Takes a key and returns a copy of the time if it is found in the table, **FAILURE** otherwise.
- **Display**: Displays to the user all info regarding a given key. Returns **SUCCESS** if data is available for the given key, **FAILURE** otherwise.