Prathyoosha Chaya

CS 163 DATA STRUCTURES

Prof. Karla Fant

**DESIGN WRITE-UP 3:**

**HEADER FILES:**

The header file will include the table class, and node struct definition, with the function prototypes of the following public/private member functions.

TABLE HEADER FILE: This header file includes the table class information along with the node structure, entry data structure (keyword, background, evolution info., candy required, additional info., list of Pokémon in area, etc.). Its private members include a pointer to the front of the hash table (array of pointers to separate LLLs), and the private utility functions to convert the key to uppercase and the hash function to find the key’s index in the hash table.

Constructor: The purpose of the constructor function is to initialize the data members. Here, the pointer to the front of the hash function and the individual pointers to each chain will be set to null.

Destructor: The destructor will deallocate all previously allocated dynamic memory - the pointer to the hash table, and the pointer to the linked lists in each element of the array (the dynamic arrays within the entry node data will be deallocated in the node destructor).

Add info function: The add info function is public and takes the key of the entry to be modified as a parameter along with the entry details to be added or changed. It returns an integer value depending on the case of the success/failure. It will first call the hash function to find the index of the key the user has provided in the hash table. Then if the first item that the index is pointing to does not match the key provided, it will traverse the chain until it is found, or return an indication of the item not being found. The contents of the node entry will be changed to match the user’s provided information.

Remove entry function: The remove entry function is public and takes no parameters. It first displays the keywords of all the entries. The user enters the keyword of the entry they would like to remove, which is then used to find the key of the item to remove by calling the hash function. Then, the LLL is traversed until it is found. If the item is found and it is the only item in the LLL (ie. no chain), then it is removed and the index pointer is set to NULL. If it is not the first item, then a temporary pointer points to the next node, and the node to be deleted is removed, and the LLL reconnected. If it is not found, nothing is removed. It returns a success or failure flag of if the item was removed/not found/not removed.

Hash function: The purpose of the private hash function is to take the key to be found/removed from the client program and find the index that it should be placed into in the hash table. These indexes generated will not be unique indexes, and the collisions will be resolved with chaining. The algorithm to create this index that will be utilized here is to add all of the characters in the key, and use the modulo operator by the array size (37 elements max) as a prime number. This will allow each key to be designated an index, without there being certain indexes that are favored by the hash function, given our data set. Before the index is found, the key is converted to uppercase using the private convert\_to\_upper function. The hash function returns the index of the key provided as a parameter by value. It is called by the add info. and remove functions.

Display all function: This public function will display all of the information in the archive (table) by going to each element in the table and traversing through its chains to display each item. This loops until the index is the array size, and the node written at the last index is the last one in the chain. This information is not in any sorted form, as the indexing function does not allow for this information to be sorted in any way understandable to the client. It returns the success/failure case (if the table is empty) by Boolean value, and it does not have any parameters.

Write function (to accompany load function): The purpose of this function is to write the contents of the hash table back into the file. It has no parameters. The entry data will be outputted to the file (no append), separated with “;” delimiters. At the end of the subject, a “\n” delimiter is placed and the index is incremented. This loops until the index is the array size, and the node written at the last index is the last one in the chain. It returns a failure or success case.

Load function: The purpose of the load function is to take the entry information in the text file and store it into the hash table. It first creates a temporary array and opens the text file to attempt to write the keyword to set EOF. While the file has not reached the end, it calls the hash function to find the correct index based on the keyword (temporary array) read, reads the rest of the entry information into the new node created in the appropriate index in the hash table. There is no need to check for a collision here, as the entry will always be added to a new node at head, and the previously added entries will trail after. It uses the existing delimiters mentioned in the “write” function to read into each entry’s fields.

Convert to uppercase function: The purpose of this private function is to convert the inputted array of characters to uppercase, using the toupper functionality in the cctype library. It takes a source and destination array of characters as input parameters, and changes the contents of destination to be the uppercase version of the source array of characters, going character by character. This will be called in the hash function.

IMPLEMENTATION FILE:

The first implementation file will contain the function definitions below.

In the main function, the menu interface will be implemented for the user to interact with the archive of Pokémon data. An object of type table “pokedex”, and an object of type entry “user\_entry” is created for the purpose of calling table class functions and passing the user’s information to be added into these functions. Then, the input file and output file variables (ifstream/ofstream type) are created, and a “menu option” variable (type int, to keep track of the menu item entered). The text file is opened, and the load function is called.

Then, the menu is outputted by calling the menu function with parameter “true”. The user can choose which option they would like to execute from the following menu format:

1: Display all information.

2: Add additional information.

3: Remove information.

4: Retrieve information.

5: Save and quit Pokedex.

6: Quit Pokedex.

The user will be prompted to enter a number, which is stored into the “menu option” variable once the function exits. If a 1 is entered, then the “display” function is entered. At this point, all data from the text file will have been read into the hash table based on its key. Once the function is exited, the menu function is entered again. The functions continue to be called until the user decides to quit (or save and quit). The program can be quit without writing to the file, or quit after writing the contents of the table to the file.

Menu function: This function takes 1 bool parameter by value (to display either the game class-related menu or the stack class-related menu) and returns an int value. It outputs the menu for the user to select, and returns the user’s selection. The menu displayed is shown in the main function.