COP 3530 – Data Structures and Algorithms I

Project 4

Due: Friday, April 1 11:59 P.M. CST (Extended Grace Period to Tuesday, April 5 11:59 P.M. CST)

Objective:

This project will build upon your knowledge of sorting and hashing. In this project, you are asked to create an abstract data type for representing a dictionary with strings as keys and integers as values. You will combine this with the usage of a stable sort to sort decks of cards. You will also need to create a makefile to compile your program. Information about creating makefiles is up on eLearning.

Problem Description:

You will be given files in the following format:

n m

suit_1

suit_2

….

suit_n

card_suit_1 card_rank_1

card_suit_2 card_rank_2

…

card_suit_m card_rank_m

The first line of the file will be two integer values, n and m. The integer n will denote the number of suits in a deck of cards. The integer m will denote the number of cards that need to be sorted.

The next n lines will be the suits of the cards that need to be sorted. Each suit will be a string. The order of the suits in the input file will indicate the sorted order for the suits. So, if spades appears on line 2 and clubs appears on line 3, spades should appear before clubs when sorted. You are asked to read each of these suits into a dictionary with the key being the suit and the value being the position of the suit in sorted order. For our example of spades and clubs, spades would have position 1, while clubs would have position 2. You will need to check the dictionary for the position while trying to sort. Note: There are alternatives ways to perform this sorting, however one of the main focuses of this project is to build and use a dictionary.

The next m lines after the suits will be cards. A card will have both a suit and a rank. The ranks are guaranteed to be integers. You will need to create a struct that represents a card and place all m cards into an m sized array to be sorted.

An example file input:

```
4 5
clubs
spades
hearts
diamonds
spades 3
hearts 9
diamonds 2
spades 2
clubs 4
```

Once this file is read in, your program should sort the cards based on suit and rank and output these cards in sorted order.

The output of your program given the example input should be:

```
clubs 4
spades 2
spades 3
hearts 9
diamonds 2
```

Creating the Dictionary

You are required to create a dictionary that allows for strings as keys and integers as values. Remember, that in C strings are represented as null-terminated char arrays (ending with '\0'). To do this, you will need to use an array of structs. Each struct composing a slot in the array should be composed of a char * for a key value and an int for the value corresponding to that key. Your dictionary should have the minimum set of functions:

- **Dictionary createDictionary(int initial_size)** – Will create a dictionary with an initial size passed in by the user. You can use n from the input file as the initial size. Each slot in the array will need to be able to hold a struct of a key and value pair. Initially, you will want each slot in your array to be set to NULL.
- **int hash(char *key, int size_of_array)** – Will take in a string and output the index in which the item should be placed. You will use a very simple hashing function that sums all the ascii values of the characters in the string and mods it by size_of_array.
- **void insert(Dictionary dict, char *key, int value)** – Will create a pair for the key and value, call the hash function, and try to place the newly created struct at the index returned by hash. If a collision occurs, then the function should perform linear probing to find the

next available slot. This may require wrapping around to the beginning of the array until an empty slot is found.

- **int get(Dictionary dict, char *key)** – Will return the value of the given key. It will first try the index returned after calling the hash function with the passed in key. If the key at that index does not match the key being searched for, starting at that index, it will iterate over the array until the key is found, then it will return the value at that position.
- **void freeDictionary(Dictionary dict)** – Will free the memory allocated for the dictionary. This should free each element in the dictionary and then the array that is storing the elements.

These are the minimum functions required by this project.

Performing the Sort

The sorting will be done with a stable sort of your choice. To perform the sorting you will need an array that represents a card. A card will contain a suit which will be a string and a rank which is guaranteed to be an integer.

You will need a single function:

- **void sortCards(Card * cards)** – This function takes in an array of cards. It then starts by sorting the array with a stable sort on the ranks. It then will sort the array again, but this time on the suits. To put the suits in the correct order, you will compare the positions of suits from the dictionary (i.e., get(suit_positions, cards[i]->suit) > get(suit_positions , cards[j]->suit)).

You must use a stable sort or else you will get the incorrect order when making multiple passes. You may include this struct and the sortCards function inside of your client. Note: I know that it is possible to perform this sorting with a single sort, however I am using this to illustrate stable sorting.

Creating Your Client

Your client should read a file passed in by the user when the program is ran. Sort the cards in the file and output the suit and value of the cards in sorted order, one card per line. You will need to create a makefile that compiles your program into an executable called cardSort.

Example of how the program should be compiled and ran:

```
…$ make

…$ ./cardSort cards.txt

clubs 4
spades 2
spades 3
hearts 9
diamonds 2
```

To help you with parsing the file, I have provided you with the scanner example on eLearning. However, do not feel that you have to use these files in the project. However, if you do, you will need to include the scanner files in your submission.

Steps for Completion:

There will be no additional tasks for this project.

1. Create the dictionary. You should be sure to test your dictionary before moving on. (30 points)
2. Create the client for your program. You should adequately test your program with multiple inputs. (50 points)
3. Provide a makefile that will properly compile your program. (20 points)

In addition, please add your name as a comment to the top of each file.

Submission Instructions:

Make sure that before submitting your program that you have tested it on the ssh server (ssh.cs.uwf.edu). There is information for how to do this on eLearning. For this project, submit a zip file containing all files required to run your project, these files will be named dictionary.c, dictionary.h, client.c, and makefile. If you used the scanner files, please include them as well. Submit your project using the eLearning dropbox. Remember that all students who turn in their projects by the due date that receive a B or better will receive an added 0.5% on their final overall average.