**Program 9 - Hash Tables**

CS 580U - Spring 2020

Due Date: 11:00 pm, May 15, 2020

*All programs will be tested on the machines in the lab. If your code does not run on the system in the lab, it is considered non-functioning EVEN IF IT RUNS ON YOUR PERSONAL COMPUTER. You can write your code anywhere, but always check that your code runs on the lab machines before submitting.*

### Driver Code and Test Input Files

### Provided Files

* + Program9.c //driver code
  + password.txt

### Grading Rubric

***TOTAL: 20 points***

## Part A & B

## Hash class uses standard static array (2 points)

## Test 1 - create empty hash (1 point)

## Test 2 - insert into a hash (5 points)

## must use open addressing to address collisions

## Test 3 - invalid password is not retrieved (2 points)

## Test 4 - verify passwords are retrievable (5 points)

## Test 5 - entries are removable (5 points)

## Style Guidelines and Memory Leaks

## *You will lose significant points for the following:*

## Makefile does not have requested format and labels (-10 points)

## Does not pass Valgrind Tests (-5 points)

## Does not follow requested program structure and submission format (-10 points)

### Guidelines

This is an individual assignment. You must do the vast majority of the work on your own. It is permissible to consult with classmates to ask general questions about the assignment, to help discover and fix specific bugs, and to talk about high level approaches in general terms. It is not permissible to give or receive answers or solution details from fellow students.

You may research online for additional resources; however, you may not use code that was written specifically *to* solve the problem you have been given, and you may not have anyone else help you write the code or solve the problem. You may use code snippets found online, providing that they are appropriately and clearly cited, within your submitted code.

*By submitting this assignment, you agree that you have followed the above guidelines regarding collaboration and research.*

For Program 9 we are going to implement a hash table for retrieving strings. You must use open addressing (linear probing) for collision resolution.

Part A: Hash Table

For Part A you are going to implement a Hash class that will store a value string and uses open addressing for collision resolution. Open addressing sacrifices insertion and lookup performance to save memory by placing the data in the next open space when there is a collision:

* struct Hash
  + char \*\* data
    - an array of strings
  + int size
    - size of the data array
  + Function Pointers
    - int insert(Hash \*, char \*)
      * returns -1 if the hash is full or the index hashed to otherwise
    - int remove(Hash \*, char \*)
      * removes the entry from the hash
      * returns -1 if the value was not found
    - char \* find(Hash \*, char \*)
      * returns the correct value based on the value parameter
      * If the value is not found, returns an empty string
    - void print(Hash \*,)
      * prints out the index and value of each element in the hash
    - int hasher(Hash \*, char \*)
      * hash function that determines an index based on the key parameter
      * You can develop whatever hash function you want, but should try to limit collisions
    - void delete(Hash \*)
      * cleans up hash memory
* Hash \* newHash(unsigned int size)
  + Should initialize an array to some static size.

Part B: Password Retrieval

Your hash should correctly store and retrieve strings along with the key. Some rules your hash must follow are:

1. You must use open addressing for collision resolution
2. Your hash table must not change size. Empty ‘slots’ are represented with NULL.

**Extra Credit (50%): Implement Channing Collision Resolution method and compare the memory and run time performances of Channing and Linear Probing techniques.**

Part C : Code Organization and Submission

* Required code organization:
  + Program9.c (driver code - You must include this file in your submission)
  + password.txt - sample text file
  + Hash.h/.c
  + makefile
    - **You must have ‘checkmem’ and ‘run’ targets**
* Use below command to tar.gz your assignment folder.
  + tar -czvf <yourid>\_program9.tar.gz <assignment folder>

**Do not include your object files or executables**

# *Expected Interface and Test Output*

### Test Commands we will run

* + make
  + make run
  + make checkmem
  + make clean
* *Sample Output:  
  TEST #1: Create a hash of the correct size*

*========================PASS========================*

*TEST #2: Insert values into the Hash*

*======== Hash Table ========*

*Index:0 Value:*

*Index:1 Value:captain.america*

*Index:2 Value:*

*Index:3 Value:*

*Index:4 Value:spiderman*

*Index:5 Value:thor*

*Index:6 Value:vision*

*Index:7 Value:blackwidow*

*Index:8 Value:hawkeye*

*Index:9 Value:wolverine*

*========================PASS========================*

*TEST #3: Verify invalid password doesn't work*

*========================PASS========================*

*TEST #4: Verify valid passwords are retrievable*

*======== Hash Table ========*

*Index:0 Value:*

*Index:1 Value:captain.america*

*Index:2 Value:*

*Index:3 Value:*

*Index:4 Value:spiderman*

*Index:5 Value:thor*

*Index:6 Value:vision*

*Index:7 Value:blackwidow*

*Index:8 Value:hawkeye*

*Index:9 Value:wolverine*

*========================PASS========================*

*TEST #5: Remove entries from the hash*

*========================PASS========================*

*All Tests Passed.*

*Don't forget to run with Valgrind and submit to Blackboard!*