DUE DATE: Friday February 5th, 2010 @ 11:59 am

# Homework Assignment 1: Memory Management

The purpose of this assignment is to introduce you to *(or remind you of)* memory management in C, including malloc(), free(), and the use of pointers.

## Description of Task

For this assignment, you will be creating a look-up table to manage 128-bit encryption keys for specific users. **These keys will be represented as 32 hex digits** (32 X 4 = 128). Each key is associated with one and only one user. The key must be represented by all 128 bits (i.e. explicit leading zeroes).

Each user is represented by its UID - which is a 16-bit identifier represented as a decimal number. Each user may have from 0 to 8 keys. UIDs may have implicit leading zeroes (i.e. 13 = 013).

The lookup table will be a 1-dimensional array of pointers.

1. It is indexed by the UID.
2. The table is created at initialization by malloc'ing the space necessary for the table.
3. Each slot should be initialized to null.
4. When a user is added to the table, a space corresponding to the size of its (MAX KEYS * KEY SIZE) is malloc'ed and a pointer to that space is placed at the appropriate location in the lookup table. MAX KEYS is 8. KEY SIZE will depend on how you represent a key (try finding a succinct representation!).

## Requirements of Task

1. Read from a file called *ValidKeys.txt* that contains rows of the following form:

   *UID KEY*

   The file is terminated by an end-of-file. Since it is assumed that you created this file, you may assume the UIDs are valid, and there are no duplicate KEYs. Initialize the look-up table and corresponding user data using this information.
2. If the file *ValidKeys.txt* does not exist, you should create it when you get the first valid ADD command. If you get a command before the first add, print out an error message and continue processing.
3. Once you have read in the input file, you should then read in a set of commands from a file called *CommandSet.txt*. The commands would be of one of the following forms.

   *ADD UID KEY*
   *DELETE UID KEY*
   *DELETE UID*
   *VALIDATE UID KEY*

*PRINT UID*
*PRINTALL*
*END*

4. You must process a command as soon as it is read until the END command is encountered.
5. You should perform all pertinent error checking (i.e. valid command, valid UID, valid key, etc.) Running edge conditions, testing for memory leaks and invalid input will be a part of the grading scripts. If you encounter an error, an appropriate error message should be written to stderr and you should continue processing. All error messages should be written to stderr.
6. Command Description:
   (a) *ADD UID KEY*
   This command should add the key associated with the user to the structure, checking to make sure the key does not already exist, that this key does not put the user over the limit of 8, etc. If the user's slot in the table is set to null, you should malloc the appropriate amount of space and set the pointer in the lookup table accordingly before adding the key.
   (b) *DELETE UID KEY*
   This command should delete the key from the user's space. If this is the only key the user had, the user space should be free'ed and the pointer to the space in the lookup table set to null.
   (c) *VALIDATE UID KEY*
   This command prints a valid or not valid message based on the existence of the requested key in the database.
   (d) *DELETE UID*
   This command should delete all keys associated with a user and free the user space associated with UID. The pointer to the space in the lookup table should be set to null.
   (e) *PRINT UID*
   This command should print all the keys associated with a user, one per line.
   (f) *PRINTALL*
   This command should print all the keys in the database - one per line. Include the associated UID with each key.
   (g) *END*
   This command should rewrite the data file ValidKeys.txt based on the current information and exit.

---

## Example:

*A ValidKeys.txt file that looks like:*
421 0123456789abcdef0123456789abcde0
422 0123456789abcdef0123456789abcde1
423 0123456789abcdef0123456789abcde2
423 0123456789abcdef0123456789abcde3

*and a CommandSet.txt file that has:*
ADD 422 0123456789abcdef0123456789abcde4
DELETE 423 0123456789abcdef0123456789abcde3
DELETE 421
VALIDATE 422 0123456789abcdef0123456789abcde4
VALIDATE 423 0123456789abcdef0123456789abcde3
PRINT 422

PRINTALL
END

*would print something like:*
The key submitted for user 422 is valid.

The key submitted for user 423 is invalid.

The keys for user 422 are:
422 0123456789abcdef0123456789abcde4
422 0123456789abcdef0123456789abcde1

The keys for all users are:
422 0123456789abcdef0123456789abcde4
422 0123456789abcdef0123456789abcde1
423 0123456789abcdef0123456789abcde2

*and result in a ValidKeys.txt file that looks like:*
422 0123456789abcdef0123456789abcde4
422 0123456789abcdef0123456789abcde1
423 0123456789abcdef0123456789abcde2

## What to Submit

Use the CS451 *checkin* program to submit a single .zip or .tar file that contains:

- All .c and .h files related to the assignment (please document your code),
- a Makefile that performs both a *make clean* as well as a *make all,*
- a README.txt file containing a description of each file and any information you feel the grader needs to grade your program.

**Filename Convention:** You can name your .c and .h files anything you want. However, the zip file and the final executable file (created by the makefile) should be named as <LastName>_<FirstName>_ASG<x>.<out/zip>. E.g. if you are John Doe and submitting for assignment 1, then the zip file should be named Doe_John_ASG1.zip and the executable should be named Doe_John_ASG1.out.

## Grading

This assignment would contribute a maximum of 5 points towards your final grade. The grading itself will be done on a 10 point scale. Following are the points break up.

1 point each for 6(b)-6(g) – total of **6 points**

**2 points** for 6(a)

**2 points** for getting the rest of the things right!

Each point may be further broken down depending on how many steps are required to achieve the task.

You may use the *ValidKeys.txt* and *CommandSet.txt* files from the example to test your program. But make sure they work for other inputs as well. **You are required to work alone on this assignment.**

---

## Late Policy

Click here for the class policy on submitting late assignments.