


# PA#2 - Concurrent Hash Table

[Start Assignment](#)

- Due Friday by 11:59pm
- Points 100
- Submitting a file upload
- File Types zip

A concurrent hash table is a data structure that allows multiple threads to perform operations on a shared collection of key-data pairs, without causing data corruption or inconsistency. The concurrent hash table consists of a linked list of nodes, each of which stores some data associated with the key. The hash value is computed by applying a hash function to the key.


Note that for this assignment, we will not be taking into consideration hash collisions. We will use [Jenkins's one at a time hash function](https://en.wikipedia.org/wiki/Jenkins_hash_function)  ([https://en.wikipedia.org/wiki/Jenkins\\_hash\\_function](https://en.wikipedia.org/wiki/Jenkins_hash_function)) which has a very low collision rate for the number of hashes we will be working with. Regardless, your test data and the final data set you will be graded against will be guaranteed collision-free.

The concurrent hash table supports the following functions:

- `insert(key, values)`: This function inserts a new key-data pair into the hash table, or updates the value of an existing key. To insert a key-data pair, the function first computes the hash value of the given name (the key). Then, it acquires the write lock that protects the list and searches the linked list for the hash. If the hash is found, it updates the value. Otherwise, it creates a new node and inserts it in the list. Finally, it releases the write lock and returns.
- `delete(key)`: This function deletes a key-data pair from the hash table, if it exists. To delete a key-data pair, the function first computes the hash value of the key and obtains a writer lock. Then it searches the linked list for the key. If the key is found, it removes the node from the list and frees the memory. Otherwise, it does nothing. Finally, it releases the write lock and returns.
- `search(key)`: This function searches for a key-data pair in the hash table and returns the value, if it exists. To search for a key-data pair, the function first computes the hash value of the key acquires a reader lock. Then, it searches the linked list for the key. If the key is found, it returns the value. Otherwise, it returns NULL. Finally, it releases the read lock and returns. The caller should then print the record or "No Record Found" if the return is NULL.

## The Hash Table Structure

```
typedef struct hash_struct
{
    uint32_t hash;
    char name[50];
    uint32_t salary;
    struct hash_struct *next;
} hashRecord;
```

Field	Description
hash	32-bit unsigned integer for the hash value produced by running the name text through the <a href="https://en.wikipedia.org/wiki/Jenkins_hash_function">Jenkins one-at-a-time function</a>  ( <a href="https://en.wikipedia.org/wiki/Jenkins_hash_function">https://en.wikipedia.org/wiki/Jenkins_hash_function</a> )
name	Arbitrary string up to 50 characters long
salary	32-bit unsigned integer (who wants a negative salary, eh?) to represent an annual salary.
next	pointer to the next node in the list



# Reader-Writer Lock Reference

The OSTEP authors have provided sample implementations that you should reference:

<https://github.com/remzi-arpacidusseau/ostep-code/tree/master/threads-sema> 

(<https://github.com/remzi-arpacidusseau/ostep-code/tree/master/threads-sema>)

**Note:**

- You will also need to grab `common.h` and `common_threads.h` from here:
  - <https://github.com/remzi-arpacidusseau/ostep-code/tree/master/include>   
(<https://github.com/remzi-arpacidusseau/ostep-code/tree/master/include>)
- Sample Reader-Writer locks implementation is here:
  - <https://github.com/remzi-arpacidusseau/ostep-code/blob/master/threads-sema/rwlock.c>   
(<https://github.com/remzi-arpacidusseau/ostep-code/blob/master/threads-sema/rwlock.c>)

# The Command File

Your program must read a text file called `commands.txt` containing some configuration information, commands, and data values. Do not read it from the console. You should hard code `"commands.txt"` into your program.

## List of Commands and their parameters

Command	Parameters	Description
threads	<number of threads>,0	The number of commands to be processed in the file. This allows you to setup a dynamic array of <code>pthread_t</code> structs. This is always the first line of the command file. Each command after this runs in a separate thread, so if you have 12 commands, this line should read <i>threads 12,0</i>
insert	<name>, <salary>	inserts or updates the data for the given name and salary value.
delete	<name>,0	Name whose record is to be deleted from the list.
search	<name>,0	Name whose record is to be retrieved and printed to the output file.
print	0,0	Print the entire contents of the list to the output file.

```
threads,11,0
insert,Richard Garriot,40000
insert,Sid Meier,50000
insert,Shigeru Miyamoto,51000
insert,Hideo Kojima,45000
insert,Gabe Newell,49000
insert,Roberta Williams,45900
insert,Carol Shaw,41000
print,0,0
search,Shigeru Miyamoto,0
delete,Sid Meier,0
delete,Hideo Kojima,0
```

Output should be written to a file called `output.txt` in the same directory as the main `chash` program and the `commands.txt` file.

Write out each command as they're executed, along with their parameters in this format:

```
INSERT,Name,Salary
```

INSERT, John Doe, 45000  
DELETE, John Doe  
SEARCH, John Doe

Write out when locks are acquired and released along with the lock type:

Examples:

```
READ LOCK ACQUIRED
READ LOCK RELEASED
WRITE LOCK ACQUIRED
WRITE LOCK RELEASED
```

### Single Record Print (e.g., from Search)

36474636, John Doe, 45000



Filename	Purpose
chash.c	Your main program that reads the commands.txt and produces output to the console
hashdb.h	Your Concurrent Hash Table struct definitions
hashdb.c	Your Concurrent Hash Table implementation, including your Jenkins function and all linked list operations
rwlocks.h	Your Reader-Writer locks struct definitions
rwlocks.c	Your Reader-Writer locks implementation
common.h	If you used OSTEP's sample code, please include it.
common_threads.h	same.
Makefile	Your Makefile that builds this project into the final executable
README.txt	Use for anything I or my graders need to know and the AI use citation (see below)

All of these should be uploaded *as a single zip file* containing all of the files.

We should be able to:

- Unzip your file.
- Run `make` to compile it into the main executable `chash`
- Your program will read `commands.txt` and then produce `output.txt`

# AI Policy

When using AI assistants, please adhere to the following policies and suggestions:

- **Put Effort into Crafting High-Quality Prompts:** Tools like ChatGPT and CoPilot, while useful, have serious limitations, and hence are often incorrect. If you provide minimum effort prompts, you will get low quality results. You will need to refine your prompts in order to get good outcomes. This will take concerted effort.
- **Be Aware of AI Limitations:** Even if you have crafted well-constructed prompts, do not blindly trust anything an AI assistant tool says or outputs. If it gives a snippet of code, number, or fact, assume that it is wrong unless you either know it to be correct or check it with another source. You are responsible for any errors or omissions provided by the tool, and these tools tend to work best for topics you fully understand.
- **Give the AI Tool Proper Attribution:** AI is a tool, and one that you must acknowledge using. Thus, you must provide these tools proper attribution if you use them for assignments. As such, you are required to provide a paragraph or two either in a comment or in the README file explaining what you used the AI for, and what prompts you used to get the results. Failure to do this will result in a violation of UCF Academic Integrity policies.

- **Know When to Use and Not Use AI Tools** Be thoughtful about when AI tools are useful and when they are not. Don't use the tool if it isn't appropriate, or if you do not have full grasp of a given concept from class.

credit to Dr. Kevin Moran for this succinct summary of AI use policy!

## Grading Rubric

100 points total

Requirement	Description	Points
Compiles	Code compiles successfully. Failure to compile results in a zero grade for the assignment.	5
Runs to Completion	Program reads in the command file and produces output. Failure to produce any output or immediate error results in zero grade for the assignment.	5
Correctly Inserts into the Hash Table	- Hash correctly computed (3 pts) - Data correctly inserted into Hash Table (2 pts)	5
Correctly Deletes Records from the Hash Table		5
Correctly Searches for Names in the Hash Table		5
Correctly Prints Single Records		5
Correctly Prints the Entire Hash Table	- Prints the table: 2 points - Prints the table in sorted order: 3 points	5
Lock Messages are Printed		5
Lock and Unlock Occur as Expected	- While writing, no other thread may read or write. (10 pts) - While reading, no other thread may write, other reads are allowed (10 pts). - No writes are allowed until all readers are complete (10 pts)	30
Final Output Matches the Test Final Output	- Correct Number of locks acquired and released (15 pts) - Correct entries in the table (15 pts): - In the correct order (5 pts) - Individual records match Test output (10 pts)	30