# CMSC 123: Data Structures

1st Semester AY 2019-2020

*Prepared by: CE Poserio & KBP Pelaez*

## [InLab] Exercise 07: HashTable ADT

### HashTable ADT

For this exercise, `HashTable ADT` is implemented with closed hashing, specifically, using double hashing as the collision resolution strategy. The list will be storing strings (technically, `char` pointers).

To create the `HashTable ADT` that will store string data, we define the following structure (defined in `hashtable.h`):

```
typedef struct hash_tag {
  uint size; // curremt number of elements stored
  uint tableSize; // size of the hash table
  STRING_ARRAY_PTR list; // array of strings
  int (*hash)(char* key); // the hash function
} HASH_TABLE;
```

Each field[1] is described by the comments in the same line. But we describe more the fifth field, `hash`. `*hash` is simply a function, which accepts a string parameter (specified by `(char* key)`) and returns an integer (specified by the data type `int`). `(*hash)` is grouped together inside a pair of parentheses to disambiguate it from declaring an integer pointer, i.e. `int *hash`. Moreover, `*hash` is a function; thus, `hash` is a pointer to a function. This is simllar if we have `int *p;` - `*p` is an `int` and `p` is a pointer to an `int`. To assign values to function pointers, we simply use function names[2] of some defined function, that matches the signature declared.

### Tasks

Implement and test the following functions (also listed in `hashtable.h`):

- `HASH_TABLE_PTR createHashTable(uint tableSize);`
- `uint isEmpty(HASH_TABLE_PTR H);`
- `uint isFull(HASH_TABLE_PTR H);`
- `void put(HASH_TABLE_PTR H, STRING key, STRING data);`
- `STRING find(HASH_TABLE_PTR H, STRING key);`
- `STRING erase(HASH_TABLE_PTR H, STRING key);`
- `void destroy(HASH_TABLE_PTR H);`
- `void printTable(HASH_TABLE_PTR H);`

These functions are also described in `hashtable.h`. The implementation for `printTable` is already in `hashtable.c`. Take note that Keys used are strings, of at most 20 characters length.

Make sure to test your implementation using the provided shell program `main.c`. The shell program accepts a simple script with the following format and commands:

1. Script must begin with a line containg to integers, the size of the table and the maximum number of elements that can be stored.
2. Succeeding lines must contain one of the following commands:
   - `+ k:<key> d:<data>` - inserts `<data>` in the hash table using the key `<key>`

[1] some fields are declared as `uint`; `uint` is just an alias for `unsigned int`; we only use nonnegative values.
[2] Function names are function pointers.

- `? <key>` - searches the node identified by `<key>` and prints the pointer value
- `- <key>` - deletes the node identified by `<key>`
- `p` - prints the hash table
- `E` - checks if the hash table is empty
- `F` - checks if the hash table is full
- `C` - deletes all nodes in the hash table, including dummy nodes

3. The last line in the file must contain the `Q` command for the program to terminate.

For all commands described, angle brackets (`<>`) are not to be written/included in the script; these are placeholders only.

### Submission

Submit your `hashtable.c` in Google Classroom.

## Questions?

If you have any questions, approach your lab instructor.