Laboratory: Hash Tables

Summary:

Prerequisites: Familiarity with structs and pointers. Familiarity with the dictionary data type.

- **Preparation**
- **Exercises**
  - Exercise 1: Unit Tests
  - Exercise 2: Overlaps
  - Exercise 3: Code Changes
- **For Those With Extra Time**
  - Extra 1: Dumping the Table
  - Extra 2: Finding Keys by Value
  - Extra 3: Deleting Elements

### Preparation

a. Create a new directory for this lab. I’d suggest Labs/HashTables, but you can choose whatever you’d like.

b. Copy the code from Examples/HashTables/KR.

c. Review the code to ensure that you understand what’s happening.

### Exercises

#### Exercise 1: Unit Tests

Right now, hash-utest.c has only one unit test. Add a variety of other tests for table_put and table_get. In particular, you should add tests that

- verify that the table works with multiple key/value pairs;
- verify that you can replace the value associated with a key; and
- verify that table_get correctly returns NULL for keys that are not there.
- verify anything else that comes to mind.

You might also find it useful to put a bunch of pairs in the dictionary. One strategy is to take a series of strings (in an array) and use the first letter as the name (key) and the full string as the value (defn).
Exercise 2: Overlaps

a. Extend the code for `install` so that it prints out a message as to where in the hash table it is putting a string. E.g.,

```c
fprintf (stderr, "%s goes in cell %d\n", name, hash (name));
```

b. Find two keys (names) that give the same index. How many words do you have to enter before you get two keys that have the same index? What does that say about the performance of hash tables?

c. Add a unit test to verify that the hash table performs correctly if we use both of those keys. In particular, you should verify that if both keys have different values, they retrieve the different values.

Exercise 3: Code Changes

As you can no doubt tell, I’ve made some changes to the K&R code. One obvious change was to separate the implementation (which relies on understanding of `struct nlist`) from the interface (which provides only two simple functions).

Find at least three other changes I’ve made to the code and consider why I might have made those changes.

*Be prepared to discuss these changes in class!*

For Those With Extra Time

Extra 1: Dumping the Table

Write a procedure, `dump (hashtable *table)`, that prints all of the key/value pairs in the table.

Note: You can’t easily unit test this procedure, so you may just want to add a dump at the end of your unit tests.

Extra 2: Finding Keys by Value

Traditionally, we use a hash table to look up values by key. But we could try the reverse, too. Write a function, `char *find_key (hashtable *table, char *value)`, that returns one key that maps to `value`. If no such key exists, return null.

Extra 3: Deleting Elements

Sketch an algorithm for deleting a key from the hash table.

Copyright © 2010 Samuel A. Rebelsky. This work is licensed under a Creative Commons Attribution-NonCommercial 2.5 License. To view a copy of this license, visit or send a letter to Creative Commons, 543 Howard Street, 5th Floor, San Francisco, California, 94105, USA.