Hash Tables

Date assigned: Wednesday, November 14, 2018
Date due: **Monday, December 3, 2018 9:15 am**
There is no late grace period for this final assignment
Points: 50

Hash Table

You are to design and implement a **Hash Table ADT**. The HT must use chaining for collision handling and rely on your **Dynamic List** to implement chaining for collision handling. The key, data pair are each void *.

The user of the HT must supply to the HT the hash function as a function pointer, a compare function as a function pointer to compare keys in the HT, a function pointer to print a key/data pair, and the HT size during the htCreate function call. You need to design the hash table header file consisting of the data structure(s) as well as the functions for the HT. The function names are to be htName, so for instance, htCreate, htTerminate, …

The HT has the expected set of functions to:

a) create a HT
b) terminate a HT
c) insert a key and associated data into the HT
d) delete a key (and associated data) from the HT returning the data
e) update a key's data
f) find a key in the HT returning the key's data
g) check if the HT is empty
h) check if the HT is full
i) print the HT
j) any other functions you so desire pertaining to a HT

In addition to implementing the data structure, you must provide a Makefile and test driver (htdriver.c that produces an executable named htdriver) that thoroughly tests your data structure. Your driver must use two hashtables, each with a key of a different data type (see Notes at the end for more instructions).

Hints:

I would make a HashTableElement similar to PriorityQueueElement which stores both the key and the data.

The hash table must accept generic data and keys (void*). This means you will need to supply to your hash table a set of function pointers to:

1) Compare two keys (see strcmp() for an example set of return values).
2) Print the key and data
3) Calculate the hash

Order Invoice Builder

Your will need to build a driver that uses at least two hashtables to solve the following problem. Example output is at the end of this document.

A company which does business internationally needs to produce invoices for their customers based on transaction records. Each transaction is recorded in the currency the purchase was made in, but the invoice needs to be displayed in US Dollars (USD). Further, the transaction records only contain product IDs and the invoice needs to list the product name and manufacturer name.

You are provided with three data files:

data/conversions.txt will contain information on how to convert from a particular currency to US Dollars (USD).

The format is:
DENOM #.##

DENOM is a three character code to denote a currency (USD, GBP: British Pound, EUR: Euro, MRD: Mars Dollars). #.## is how many US Dollars 1 DENOM is worth.

data/items.txt will contain information about various items for sale.

The format is:
ID# Name Manufacturer

ID# is any positive int, Name and Manufacturer are each at most 9 printable characters.

data/actions.txt will list various purchases for one customer.
ID# Quantity Cost DENOM

ID# is the item purchased, Quantity is the number of items purchased, Cost is the cost of one item, and DENOM specifies the currency of Cost.

Your task:

For each line in data/actions.txt print the following to the file data/invoice.txt:

ID# Name, Manufacturer Quantity Cost_In_USD_For_1_Item Total_Cost_in_USD

For the dollar amounts, use printf() output formatting to set the precision to two digits past the decimal point.

Notes:

0. You must use at least two hash tables to solve this problem, each using a different datatype for the key.

If a string is used for a key to the hash table: be sure to make all keys (strings) the same length (pad any small strings with '\0' at the end). You must use the string hash function presented in class ("*31"), mod the size of the table.

If an int is the key, use the midsquare algorithm, taking the middle 8 bits, mod the size of the table.

There are a number of structs to build in this assignment. Statically allocate the strings inside the structs.

Page 36 in your book has a discussion about function pointers.

1. You will need at least two modules other than your list module. You will need a hash table project (**HashTable**) and an Invoice project (**Invoice**) for sure.

2. Submit a file called **cs300_7_PUNetID.tar.gz** by 9:15am on the due date. This file must include at least the two new projects you are to write as well as **GenericDynamicList**. Each project is to be complete such that I can type make in any of the projects and execute any driver I so desire. Make sure you have all dependencies set correctly and that each Makefile builds the appropriate object files before building the executable. Also, each module's Makefile must have a target called **valgrind.**

I will clean all projects BEFORE typing make in the Invoice project.

Turn in a color, double sided, stapled packet of code by the due date. The packet must be in the following order:

- invoice.c (.h then .c if you have both, otherwise just .c)
- Makefile for invoice
- ht.c (.h then .c)
- htdriver.c
- Makefile for HashTable
- Any extra .h/.c pairs you have. (do not include any code from previous projects)
- Extra Makefiles printed in any order

Sample Input/Output

converstions.txt

```
USD 1.0
CAN 0.75
YEN 0.20
GBP 1.75
EUR 1.90
MRD 0.34
```

items.txt

```
1 i7CPU Intel
2 i5CPU Intel
3 i3CPU Intel
18 iMac Apple
19 MacBook Apple
20 MacBookPro Apple
10202 9mmFan Corsair
20102 CPUFan Corsair
```

actions.txt

```
1 1 500 USD
2 16 270 GBP
10202 9 150 YEN
```

invoice.txt

```
1 i7CPU, Intel 1 500.00 500.00
2 i5CPU, Intel 16 472.50 7560.00
10202 9mmFan, Corsair 9 30.00 270.00
```