Homework 3

100 Points

BINARY FILES and STRINGS

Grading Hw_3_A (20Points)

There are a number of errors (about 6) in this program.
Locate all errors, fix them run the program and save its output.

Grading Hw_3_B (80Points)

Project: Hardware Store Database – hashing to disk // See next page

The program passes the input filenames to the program as an argv parameter.

1. Use of the argv parameter//file name	– 10
2. Create binary file	– 10
3. Insert data from an input text file	– 10
4. Insert data from keyboard	– 20
5. Delete	– 10
6. Find and display	– 10
7. Error checking (advanced string manipulation)	– 10

For this assignment you will need two input text files: Copy and paste the following data to two input text files:

hardware.txt

```
6745,MOLLY BOLT:57
5675,SCREW DRIVER:199
1235,WIDGET:28
2341,WING NUT:89
8624,SLEDGE HAMMER:27
9162,FLASH LIGHT:25
7146,CEMENT BAGS:113
2358,VISE:44
1622,HAMMER:15
1832,THERMOSTAT:78
3271,NAIL:2345
4717,BRACE:234
```

Next Page

more_hardware.txt

```
9524,CLAMP:523
1524,SANDER:99
5219,SAW:211
6275,SAW BLADE:675
5392,BOLT (REGULAR):1311
5192,SCREW DRIVER:789
```

Hardware Store Database

Write a program that that allows additions to, deletions from, or displays of database records in a hardware store database. The records in the database will consists of the following fields:
id – string length is exactly 4 characters (digits only) // key field
name – string length up to 20 characters (letters, spaces, and ( )). This string represents the name of a product that could be found at a hardware store.
qty – an integer. The number represents the quantity available for that product.

This challenging project involves more than 400 lines of code and uses everything from this chapter. It uses strtok() and strtod() and possibly other string.h functions.

Your program must:

a. Create the empty hashed file by writing an empty hash table repeatedly to a new file. Pretend this is a real-world application that might involve millions of records, therefore hashing to memory is not an option. For this assignment the empty hashed file will hold 30 three-struct buckets – no overflow area. Assume (pretend) that the empty hashed table could store only 10 three-struct buckets. To create the file you will have to write this table 3 times to the hashed file.
b. Use **`hardware.txt`** as a default input file name. If the user passes the input file's name to the program as an argv parameter, use that name. Assume that the input file is valid (has valid data in valid format). Then you will do batch insertions into the hashed file: read one item from the input file into a HARDWARE struct, hash the key (unique id) and add the record to the file, and so on, one record at a time.
c. Treat overflows in a bucket as an un-resolvable problem but report the error: report and reject record additions that would overflow the 3-record bucket for each hash location. Save all rejected items to an output text file named **`rejected.txt`**
d. The hash key is the id and you should sum the cube of the ASCII values of the characters making up the id before dividing by 30 to find the bucket to put the record.

Next Page

e. Put names in the file in one case of the other, but not mixed. For instance `Bolt` will be stored as `bolt` or `BOLT`.
f. Present the user with a menu to
    a. Search (by the unique key, id)
    b. Delete (by the unique key, id)
    c. Allow more data to be added from a specified input text file to your data base. First check the argument list for the next input file name. If not found or there's nothing left in that list, prompt the user to enter another file name.
    d. Insert one record from the keyboard. Do complete error checking for keyboard added records: the user should enter id, name, and quantity on one line with any amount of whitespace around the three tokens. It is a major task to parse, case, and space the requested input so that it agrees with your format in the hashed file. A valid name should consist of letters, spaces, and (). You must follow all the rules for good token parsing.
    e. Exit

Run the program at least twice (one session is for error testing). Duplicate id not allowed.

NOTE: Please review the class examples before you start working on this assignment:

`e_5_2_hash.c`
`e_5_7_hash_ovfl.c`

You are expected to write similar code. Reuse as much code as possible.