CIS 26B
Advanced C
Programming Assignments

Homework 3

Binary Files and Strings

100 Points

Projects

`26B_H_3A.c` – Basic Binary Files Operations: There are a number of errors (about 6) in this program. Locate all errors, fix them, run the program, and save its output as a comment at the end of the source file.

`26B_H_3B.c` – Hardware Store Database – hashing to disk (next pages).

Review the following two examples. You are expected to write similar code.
Reuse as much code as possible.

Example 5-2: (**`e_5_2_hash.c`**)
- Create an empty hash table of "struct record"
- Prompt the user to enter data, hash the key, and insert the data at the index given by the hash function, and repeat, to populate the hash table
- Write the hash table to a binary file: we are "cheating" here because we write the hashed table to the file and then pretend that it does not exist so we could demonstrate how to use hash search in a binary file
- Collisions are not solved in this example

Example 5-7: (**`e_5_7_hash.c`**)
- A record in the binary file is a "bucket" of 4 structs
- This provides room for synonyms
- When a bucket becomes full, the records are added at the end of the file in an overflow area.

Grading

Program 3A	– 20Points
Program 3B	– 75 Points

1. File names	– 5
2. Create empty binary file	– 10
3. Insert from the keyboard	– 15
4. Insert from text file	– 10
5. Find and display	– 15
6. Delete	– 10
7. Error checking (advanced string manipulation)	– 10

Self-Assessment Report: – 5Points

Write a short report (**`26B_H3Report.docx`**), briefly explaining your code and containing an assessment of your implementation based on the above grading criteria.

Hardware Store Database – hashing to disk

Write a program 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.

Assume this program deals with a large number of records, therefore hashing to memory is not an option. Assume that an empty hashed table's capacity is 10 three-struct buckets.

Use hardware.dat as a default file name. If the user passes a file name to the program as an argv parameter, use that name.

Open the file for writing to/reading from (in binary). If the file is not found, create an empty hash table of 10 three-struct buckets and use it to create an empty binary file to be used for hashing and searching. The empty hashed file will hold 30 three-struct buckets and no overflow area (i.e. write the empty hash table to the file 3 times).

Report and reject record additions that overflow the 3-record bucket for each hash location.

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.

Put names in the file in upper case. For instance, `Bolt` will be stored as `BOLT`.

Present the user with a menu to:

- S - Search (by the unique key, id)
- D - Delete (by the unique key, id)
- I - Insert one record from the keyboard.
- B - Allow more data to be added from a specified input text file to your data base. For testing use the following two text files: jan_hw.txt and new_hw.txt
- E - Exit

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.

Examples of valid input:

```
9162,Flash Light:25
   9162   ,      FLaSH      LIghT :    25
```

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.

Examples of invalid input:

```
9162 Flash Light:25         // missing ',' after ID
9162,Flash Light 25         // missing ':' after name
91A2 Flash Light:25         // invalid ID – digits only
91-2 Flash Light:25         // invalid ID – digits only
5392,BOLT [REGULAR]:1311    // invalid name – [ ] not allowed
5392,BOLT (REGU1AR):1311    // invalid name – digits not allowed
9162,Flash Light:2A         // invalid quantity
9162,Flash Light:           // incomplete input line
```

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.

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

When reading data from a file, we assume data have been validated, therefore we may consider that the file is valid and correctly formatted.

jan_hw.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
```

new_hw.txt

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