CSCI 352 Unix Software Development
Summer 2016
Assignment 1

Submitting Your Work

This assignment is worth 15% of the grade for the course. Save your program files (including header files and makefile) in a zipped tar file and submit that file via the **Assignment 1 Submission** item on the course web site. You must submit your assignment by 11:00am on Monday, July 18.

Your assignments will be evaluated on correct functionality and conformance to the coding standards described at the end of this assignment specification.

Outfitters 352

With the aim of boosting the retirement fund of a certain, rapidly-aging CS professor, students registered in CSCI 352 are donating their labor (in exchange for course credit) in the development of an on-line shopping system, to be known as "Outfitters 352". Members of the class of Summer 2016 have the privilege of working on the product inventory system which is to use a light-weight, rapid-access file that will provide excellent performance, without all the overheads of a database system. This is to be done by implementing a *hashfile*, which is a hash table, with direct chaining, stored in a binary, random-access file.

Previous development teams have made some progress in developing modules for products and prices. Those modules are available for your use with the hashfile.

Inventory items

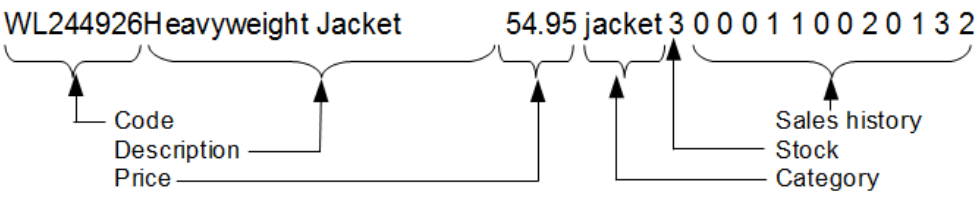
For each product, the inventory system keeps track of:

- Product code: a string of 8 characters
- Product description: a string of 31 characters
- Price: in dollars, accurate to the nearest cent, for example 23.45
- Product category: one of jacket, shoes, shirt, sweater, pants, accessory
- Stock on hand: an integer number (which may be negative)
- Sales history: number of items sold in each of the preceding 12 months. The first number in this sequence represents the total sales quantity for the most recent month.

Inventory File

The current state of the product inventory is saved in the file `inventory.txt`. Each line in that file represents one product. The values of the data components for that product are listed on the line in the same order as the components are listed in the above description of the inventory items.

For example, the first line in the file is



Transactions

Transactions are events which cause a change to the inventory data. There are five types of transaction in this system:

- Sale: a quantity of one product has been sold and is removed from the stock for that product.
- Delivery: a quantity of one product has been delivered and is added to the stock for that product.
- New product: a new product is added to the inventory.
- Delete product: an existing product is deleted completely from the inventory.
- Price change: the price of one product is changed.

Transaction File

The transactions for the last month are provided in the file `transactions.txt`. Each line in that file gives the data for one transaction. There may be more than one transaction for any product. The data given for a transaction depends on the type of transaction.

- Sale: product code and quantity sold. For example:
  `SALE WL237870 2`
  This represents a sale of 2 units of the product with code WL237870.
- Delivery: product code and quantity delivered. For example:
  `DELIVERY WL242373 10`
  This represents a delivery of 10 units of the product with code WL242373.
- New product: product code, description, price and category of a new product. For example:
  `NEWPRODUCT WL220312New Wave Jacket 55.50 jacket`
  This represents a new product with product code WL220312, description "New Wave Jacket" (padded out with spaces to 31 characters), price $55.50, and category jacket.
- Delete a product: product code. For example:
  `DELETE WL036147`
  This means that the product with code WL036147 is to be deleted from the inventory.
- Price change: product code and new price. For example:
  `PRICE WL162343 45.50`
  This indicates that the price for the product with code WL162343 is to be changed to $45.50.

Your Task

Your task is to write four program modules:

1. `hashfile.c`, which implements all the functions listed in the provided header file `hashfile.h`.

2. `tokenizer.c`, which implements the two functions listed in the provided header file `tokenizer.h`.

3. One program to build a new inventory hashfile from the products listed in the provided data file `inventory.txt`.

   Use the `display_file()` function to prove that all inventory items have been correctly added to the hashfile.

   This program must get the name of the inventory text file and the name of the resulting hashfile from its command line arguments. For example:

   ```
   ./your_prog1 inventory.txt hashed.txt
   ```

   where `inventory.txt` is the inventory text file and `hashed.txt` is the resulting hashfile. Feel free to use a better name than `your_prog1`.

4. Another program to update an existing inventory hashfile by applying the transactions listed in the provided data file `transactions.txt`.

   Use the `display_file()` function to prove that all transactions have been correctly applied to the hashfile.

   This program must get the name of the transaction file and the name of the existing hashfile from its command line arguments. For example:

   ```
   ./your_prog2 transactions.txt hashed.txt
   ```

   where `transactions.txt` is the inventory text file and `hashed.txt` is the existing hashfile (previous created by `your_prog1`). Feel free to use a better name than `your_prog2`.

   The following rules apply to the transactions:

   a) There may be zero, one or more than one transaction for each product.

   b) For a new product, the stock is to be set to zero and the sales history is to be set to zero for each month.

   c) The total sales quantity for the month (perhaps from multiple sales transactions) is to be added as the first component of the sales history, pushing all the existing history items along one place and discarding the last number from the sequence.

   d) For any product for which there are no sales transactions in the transactions file, assume zero sales for the month and update the sales history accordingly.

   e) Check for errors in the transactions. If any error is found, display a message on standard output with the details of the transaction and the nature of the error. A transaction containing any error is not to be used to update the inventory data.

      The type of error to look for depends on the type of transaction.

- For sale, delivery, delete and price change transactions, report an error if the transaction product code does not exist in the inventory.
- For a new product, report an error if a product with that code already exists in the inventory.

f) A sale transaction may be for a quantity larger than the current inventory of that product. This is quite normal, as the stock level, as recorded in the inventory actually means "uncommitted stock" and a negative value just acts as a flag for our purchasing department.

Saving your files in a zipped tar file

You need to submit all your C and header files. First you need to bundle them up into a single *tar* file. The term "tar" is an abbreviation of "tape archive" and goes back to the days when people would save a back-up copy of their files on magnetic tape. Nowadays, with the price of large disk drives so low, nobody uses tape anymore, but the concept of tar files survives.

1. Use the command:

   ```
   tar -cf Assg1.tar *.c *.h makefile
   ```

   The -cf specifies two options for the tar command: 'c' means create and 'f' means that the name of the resulting tar file comes next in the command.

   Following the name of the tar file, we list the files to be included in the tar file. In this case, we want all the files in your current directory whose names end with `".c" or ".h"`.

2. If you now use the command `ls` you should now see the file Assg1.tar in your directory.
3. If you use the command

   ```
   tar -tf Assg1.tar
   ```

   it will list the files within the tar file.

4. Now compress the tar file using the gzip program:

   ```
   gzip Assg1.tar
   ```

5. By using the `ls` command again, you should see the file Assg1.tar.gz in your directory. This is the file that you need to submit through the **Assignment 1 Submission** link in the course web site.

Coding Standards

1. Use meaningful names that give the reader a clue as to the purpose of the thing being named.
2. Avoid the repeated use of numeric constants. For any numeric constants used in your program, use #define preprocessor directives to define a macro name and then use that name wherever the value is needed.
3. Use comments at the start of the program to identify the purpose of the program, the author and the date written.
4. Use comments at the start of each function to describe the purpose of the function, the purpose of each parameter to the function, and the return value from the function.
5. Use comments at the start of each section of the program to explain what that part of the program does.
6. Use consistent indentation.