

Library DB: Binary File I/O, Hash and Inverted Table Indexing

For this project you provide indexing capabilities for a simple database file storing information about a collection of books. The database will consist of a sequence of logical records, of varying sizes. Each record will contain multiple fields (data values), some of which will be unique to that record and some of which may be duplicated in other records. Logically, the database file will never contain two copies of the same record.

Logically, each record in the database will be of the following form:

ISBN:	0-345-43481-1
Title:	The Last Full Measure
Author:	Shaara, Jeff
Category:	FICTION
Year:	1998
Sales:	567883

The ISBN is always a string of 13 characters. The title and author name are strings of arbitrary length. The year of publication and number of copies sold are integers. The category will be one of the labels given in the table below. When writing a book record to the database file, each category will be represented by the integer value shown in the table.

Category Label	DB representation
UNKNOWN	0
FICTION	1
NONFICTION	2
SIFI	3
FANTASY	4
REFERENCE	5

The database file will be a binary file, whose records are stored in essentially random order. Physically, each record will be organized as follows:

# of bytes	value	type
4	length of record (excluding these 4 bytes)	int
13	ISBN	char
2	book category flag	short
4	sales	int
4	year published	int
varies	book title	char
1	separator symbol (':')	char
varies	author name	char

You will build two indices for the database file. The primary key index will be a hash table, using the **ISBN** field as its key. Names are guaranteed to be unique (i.e., no two different records will contain the same name string). Hash table details are given below. The second index will be an inverted table (section 9.3.3 of Kruse/Ryba and the course notes), using the **author name** as a key value. It is certainly possible that two different records will contain the same **author name**. Details of the inverted table are also given below.

Each slot of the hash table will store a primary key value and the address (byte offset) of the corresponding record in the database file. The address of a record is defined to be the byte offset to the first byte of the file (i.e., **the first byte of the record length in this case**). Note that byte counts start at 0.

Each slot of the inverted table index will store a secondary key value and corresponding primary key value.

Given the byte offset to a record, it is easy to advance the read pointer (read point from the file stream) or the write pointer to that offset and then or write the record. You will want to make use of the input stream functions `seekg()` and `tellg()`, and the corresponding output stream functions `seekp()` and `tellp()` to move the read and write pointers.

Since this project requires performing interleaved reads and writes on the same file, you should use an `fstream` to access the file, rather than separate `ifstream` and `ofstream` objects. There's not much difference between the two, but it is somewhat dangerous to access a file concurrently via two or more different streams.

Note: the complete list of database records is NOT to be stored in memory. Individual records may be read from disk to memory as needed.

On startup, the program will read the database file and build the specified indices. Next, the program will read a command file and carry out the commands, writing any output to a log file.

Data Structures:

The primary data structures of this project are a hash table and an inverted table. Your implementation is under the following specific requirements:

- You must encapsulate the two indices as C++ classes. The use of templates is preferred but not required (i.e., no penalty).
- The data that is stored in each index slot must be encapsulated in a C++ class.
- Given the nature of this project, no two distinct database records will contain duplicate names. Your implementation should detect the insertion of a duplicate name and reject the insertion.
- For testing, both of your indices should have the ability to display themselves to a specified output stream.

Since the index slot objects are **manipulated** only within the indices, it is acceptable for the data members of the slot objects to be public. Otherwise, data members of classes should be private.

The specification implies additional classes that you should identify and implement.

Hash Table Index:

The hash table in this project does NOT store the data records. Rather, each slot stores a hash record containing a key value (**ISBN** string) and the address of the corresponding record in the database file.

When a lookup is performed, the hash table should be passed a key value. A successful lookup should result in the hash table returning the corresponding record address; a failed lookup should result in the hash table returning a logically invalid record address.

The hash table should use the `ELFhash` function described in the course notes, and linear probing. Since linear probing is used, there is no reason to worry about failed insertions unless the table is completely full. The condition for terminating a probe should be if the number of slots examined reaches the size of the hash table.

The number of slots in the hash table should be two times the number of records in the database file.

Inverted Table Index:

The inverted table contains an array of table records. Each table record contains a secondary key value (author name) and the corresponding primary key value. The array of records should be sorted by key value, so that binary search can be used. The inverted table should be initialized to contain the same number of slots as the hash table.

Insertions to the inverted table may require shifting entries. However, if the inverted table is full then insertions will simply fail. Records may also be deleted. In that case, the corresponding entry in the inverted table should be deleted as well, requiring a shift of part of the array.

Program Invocation:

Your program **must** take the names of the input and output files from the command line — failure to do this will irritate the person for whom you will demo your project. The program will be invoked as:

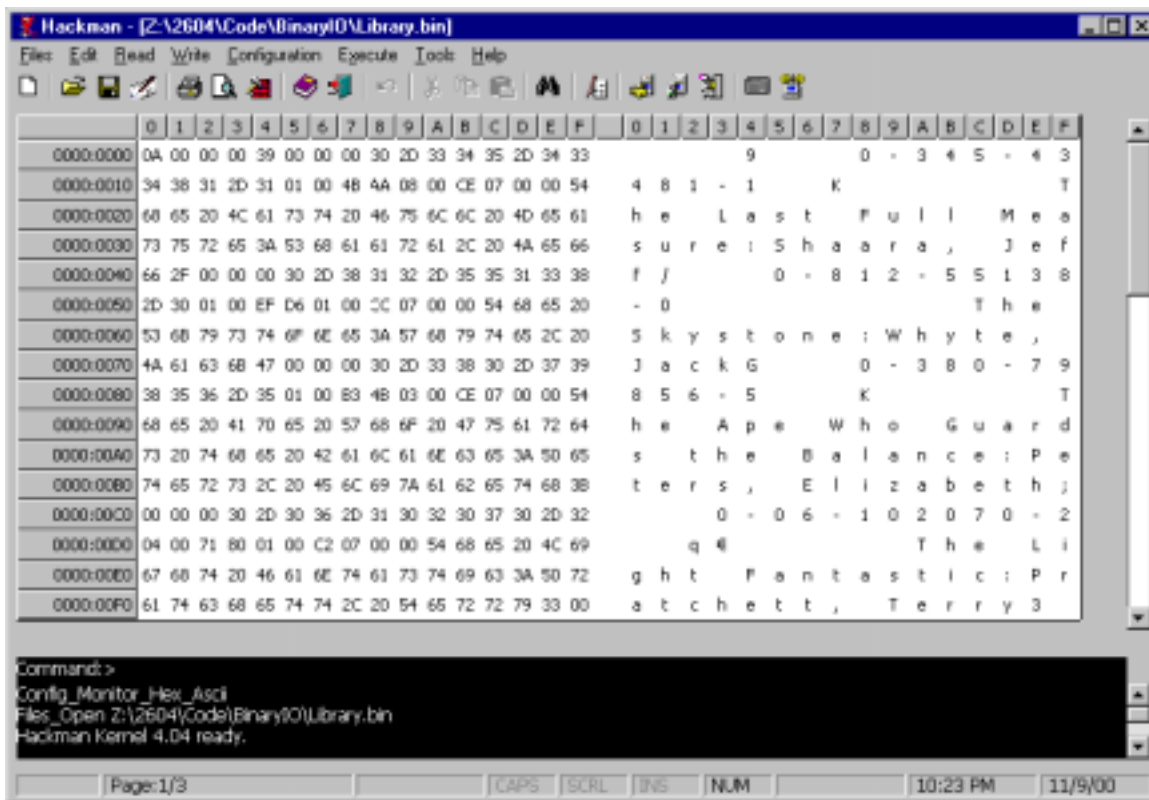
```
index <database file name> <command file name> <log file name>
```

If either of the specified input files does not exist, the program should print an appropriate error message and either exit or prompt the use for a correction.

Database File Description:

The database file will begin with an integer specifying the number of records (this value is used to determine the size of the hash table and the inverted table). The remainder of the database file consists of a sequence of records (as described earlier).

The database file is guaranteed to conform to the specified syntax. Here is a hex view of a sample database file:



Here is part of an ASCII dump of the records in the same database file:

```

Offset: 4
Length: 57
ISBN: 0-345-43481-1
Title: The Last Full Measure
Author: Shaara, Jeff
Category: FICTION
Year: 1998
Sales: 567883
Offset: 65
Length: 47
ISBN: 0-812-55138-0
Title: The Skystone
Author: Whyte, Jack
Category: FICTION
Year: 1996
Sales: 120559
Offset: 116
Length: 71
ISBN: 0-380-79856-5
Title: The Ape Who Guards the Balance
Author: Peters, Elizabeth
Category: FICTION
Year: 1998
Sales: 215987
Offset: 191
Length: 59
ISBN: 0-06-102070-2
Title: The Light Fantastic
Author: Pratchett, Terry
Category: FANTASY
Year: 1986
Sales: 98417
Offset: 254
Length: 51
ISBN: 0-06-102069-9
Title: Equal Rites
Author: Pratchett, Terry
Category: FANTASY
Year: 1987
Sales: 99432
. . .
Offset: 492
Length: 51
ISBN: 0-553-56273-8
Title: Doomsday Book
Author: Willis, Connie
Category: SCIFI
Year: 1992
Sales: 187403
Offset: 547
Length: 63
ISBN: 0-553-57538-4
Title: To Say Nothing of the Dog
Author: Willis, Connie
Category: SCIFI
Year: 1998
Sales: 174350
    
```

Note: the length values do not include the 4 bytes in which the length field is stored.

The offsets shown are correct.

The category flags have been mapped to the appropriate labels.

It's a good idea to implement code to produce something like this output for the given database files.

With this, you can easily verify that you are reading data correctly from the binary file, and also that your updates to the binary file are correct.

Driver File Description:

Lines beginning with a semicolon (` ; `) character are comments; your program will ignore comments. An arbitrary number of comment lines may occur in the input file.

Each non-comment line of the command file will specify one of the commands described below. Each line consists of a sequence of “tokens” which will be separated by single tab characters. A newline character will immediately follow the final “token” on each line.

isbn <isbn>

This causes the hash table to be searched for the location of the record, if any, that contains the given ISBN string. After the search, a message should be printed to the log file indicating either that the search failed, or printing the address of the record followed by the formatted and labeled contents of the record (all fields).

author <author name>

This has the same effect as the previous command except that the inverted table is searched for the address(es) of all matching records. There may well be multiple matches to the given author name. If that happens, you should report each match (formatted as for the **isbn** command). An error message should be logged if no matches are found.

add <isbn> <category label> <title> <author name> <sales> <year>

If a record containing the given ISBN is already in the database file, do not change the database. If either index is full, do not change the database. Otherwise, this causes the updating of both indices and the insertion of a new record at the end of the database file (just move to the end and start writing). ~~Don't forget to update the counter at the beginning of the database file as well.~~ Log an informative message in all cases.

remove <isbn>

This causes the hash table to be searched for the location of the matching record, if any. If a match is found, then the corresponding record should be marked with a tombstone. The inverted table must also be updated to reflect the deletion of this record. The record should NOT be physically deleted from the file, or overwritten.

A message confirming the deletion should be logged, including the address and contents of the deleted record. If no match is found, a failure message should be logged.

change <isbn> <units sold>

Assuming there is a record containing the specified ISBN, its sales field is replaced with the given sales value. A success message, including the record address and contents, or a failure message, should be logged.

hashtable

Causes a formatted display of the hash table to be written to the log file. The display should list each cell of the table, indicating that the cell is empty, or is a tombstone, or showing the key value and address stored there.

invertedtable

Causes a formatted display of the inverted table to be written to the log file. The display should list each cell of the table, indicating that the cell is empty, or is a tombstone, or showing the primary and secondary key values stored there.

exit

This causes the indexing application to terminate. The command file is guaranteed to end with an exit command.

You may assume that the command file will conform to the given syntax, so syntactic error checking is not required. If an error occurs during the parsing of the command file, there's an error in your code. However, your program should still attempt to recover, by "flushing" the current command and proceeding to the next input line.

A sample command file, and corresponding log file, will be posted shortly. In the meantime, you should be able to easily construct one for the given database file.

Log File Description:

Since this assignment will be graded by TAs, rather than the Curator, the format of the output is left up to you. Of course, your output should be clear, concise, well labeled, and correct. The first two lines should contain your name, course (CS 2604), and project title.

After the indices are built from the database file, you should dump the contents of each index (formatted nicely) to the log file.

The remainder of the log file output should come directly from your processing of the command file. You are required to echo each command that you process to the log file so that it's easy to determine which command each section of your output corresponds to.

Submitting Your Program:

You will submit a zipped file containing your project to the Curator System (read the *Student Guide*), and it will be archived until you demo it for one of the GTAs. Instructions for submitting are contained in the *Student Guide*. You will find a list of the required contents for the zipped file on the course website. Follow the instructions there carefully; it is very common for students to suffer a loss of points (often major) because they failed to include the specified items.

Be very careful to include all the necessary source code files. It is amazingly common for students to omit required header or cpp files. In such a case, it is obviously impossible to perform a test of the submitted program unless the student is allowed to supply the missing files. When that happens, to be fair to other students, we must assess the late penalty that would apply at the time of the demo.

You will be allowed up to five submissions for this assignment, in case you need to correct mistakes. Test your program thoroughly before submitting it. If you discover an error you may fix it and make another submission. Your last submission will be graded, so fixing an error after the due date will result in a late penalty.

The submission client can be found at: <http://spasm.cs.vt.edu:8080/curator/>

Programming Standards:

The GTAs will be carefully evaluating your source code on this assignment for programming style, so you should observe good practice. See the Programming Standards page on the course website for specific requirements that should be observed in this course.

Evaluation:

You will schedule a demo with your assigned GTA. At the demo, you will download your submitted project, perform a build, and run your program on the supplied test data. The GTA will evaluate the correctness of your results. In addition, the GTA will evaluate your project for good internal documentation and software engineering practice.

Here is an estimate of how much weight will be given to each of the factors that the GTA will consider may be added here later.

Pledge:

Each of your program submissions must be pledged to conform to the Honor Code requirements for this course. Specifically, you **must** include the pledge statement provided with the earlier project specifications in the header comment for your main source code file.