University of Toronto

CSCB28F—File Structures, Fall 1998

Assignment 1: Cosequential Processing

Due: Wednesday 21 October at 10:00 am.
Partners: not allowed for this assignment.
Worth: 15%.

Introduction

The purpose of this assignment is to teach you cosequential processing. It will help everyone get comfortable with file access in C. To learn about this general technique, you should read your lecture notes on the topic and Sections 7.1 and 7.2 of the textbook.

You are to write a program that maintains information on books in the warehouse of the University of Toronto bookstore. We will assume on this assignment that each bookstore (St. George, Scarborough and Erindale) prepares a transaction file every day. Each transaction record makes requests of the following type:

- send *xxx* copies of book *yyy*.

In addition to these 3 transaction files, there is another daily transaction file internal to the warehouse. The transaction records of this file make requests of the following types:

- add new book *yyy*;
- delete book *yyy*;
- change information of book *yyy*; and
- list all information of book *yyy*.

The information is maintained in a master file ordered by the field *Identifier* that uniquely determines each book. Every night, the four transaction files introduce modifications to the master file. Your task is to process all transactions contained in the different transaction files updating the master file. You can read the master file *only once*. Consequently, you must process all transaction files *at the same time*. In addition, changes in a record of the master file must be recorded only once, after processing all transactions for certain book. You can assume when testing your program that there are at most 4 transaction files. However, your implementation must be *independent of the number of transaction files* involved. We will also assume that the records on each transaction file are ordered by the field *Identifier*.

File Descriptions

The files contain one record per book. The records are of fixed length, with fixed-length fields, and they are sorted according to *Identifier*.

The records in transaction files contain the same information as in the master file with the exception that there is one more field *Type*. The action you should take depends on the contents of this field:

- Type=1: insert a new record into the master file;
- Type=2: delete an existing record from the master file;

- Type=3: change the information of an existing record in the master file;
- Type=4: list the information of the book *Identifier* in the master file; and
- Type=5: send *xxx* copies of book *yyy*.

Field#	**Field Name**	**Type**	**Length in bytes**	**Notes**	**File**
1	Identifier	numeric	long	unique key	master/trans
2	ISBNNumber	numeric	int		master/trans
3	Author	alphabetic	char[20]		master/trans
4	Title	alphabetic	char[30]		master/trans
5	Editorial	alphabetic	char[30]		master/trans
6	ListEdCities	array of string	char[4][15]		master/trans
7	TimePubl	alphanumeric	char[5]	format = mm/yy	master/trans
8	NumbCopies	numeric	int		master/trans
9	Threshold	numeric	int		master/trans
10	Type	numeric	int		trans

Your program

First, you must create the master file and the transaction files. Call these files: `master`, `t0`, `t1`, `t2`, `t3`. Second, your program should process the transaction files sequentially and in batch mode (no further input from the user while it executes). Call your executable program `books` and your updated master file `newmaster`. We will use the following command to run your program:

```
books master t0 t1 t2 t3 newmaster newrequests error.log
```

Do not read the entire transaction files or the master file all at once into main memory. In real situations, these files may be too large to fit in main memory.

There may be errors in the transaction files and your program should do some error checking. Do not apply the transactions with errors to the master file. Instead, just output the transaction and a short message to an error file. There are two kind of errors: (a) errors that can be detected by looking at the transaction record (bad syntax), and (b) errors that depend on the present content of the master file (for instance, a change in a nonexisting book). Every transaction from a bookstore reduces the number of copies in the warehouse. When the number of copies of a book is less than its threshold, create a record in a file `newrequests` containing only the identifier of the book.

How to hand in your assignment

First, you must submit **all** your files. This involves your program `books.c` (the program that process the transactions and updates the master file), and all your files including the transaction files from your tests, the error output `error.log`, and `newmaster` and `newrequests` created by your tests.

The marker must be able to compile your program using **gcc**, and run it successfully on an appropriate test file. Use the submit command to submit copies of your source codes:

```
submit -N a1 cscb28f *
```

At the deadline, you must hand in source listing for your C programs (Unix). You do not need to include a report but marks will be given for good style and modularity. The program should be properly documented with features such as comments for each variable and function, variables with sensible names, and so on. In addition, your tests should contain comments indicating what each test is checking. Also include a description of the transaction error checking done by your program.