University of Toronto

CSC228F—File Structures, Fall 1997

Assignment 2: On line processing

Due: Tuesday 4 November; electronic submission at 6:00 pm, and source listings at 6:10 pm.
Partners: you may complete and submit this assignment with a single partner from the class. Both partners will receive the same mark.
Worth: 15%.

Introduction

The purpose of this assignment is to teach you external sort, differential files, and direct access using hashing methods. To learn about these techniques, you should read your lecture and tutorial notes on the topic and Chapter 10 of the textbook.

As in assignment 1, you are to write programs that maintain a set of flights where the users will make requests of the following types: add a new flight; delete a flight; and change information of a flight.
The information is maintained in a master file ordered by the field *Identifier* that uniquely determines each flight. There is only one transaction file that introduces modifications to the master file. Your task is to process all transactions contained in the transaction file updating the master file. In assignment 1, we have considered the unrealistic assumption that the transaction records were ordered by the field *Identifier*. One of the purposes of this assignment is to teach you how to deal with unordered transactions.

Part A: External sorting

The files contain one record per flight with the same description as in assignment 1. Your task in this first part is to sort the transaction file. You are to use some external sorting method. Of course, you are NOT allowed to read the the entire transaction file into main memory. Instead, you must use any of the external sorting methods in Chapter 7 of the textbook (see also your tutorial notes).

Part B: On line processing with binary search

We now assume that the transactions are not ordered by identifier and they are to be processed against the master file as they arrive. Change in the information of a flight and deletions of flights are applied directly to the master file. Use binary search for finding the appropriate record in the master file.
Insertions, on the other hand, are treated with a *differential file* since otherwise will be very costly to maintain the master file after each insertion. The differential file is unsorted and contains one record per insertion transaction. When a change or delete transaction arrives, first search the differential file using linear search. If the identifier searched for is not in the differential file, then search the master file using binary search. You do NOT need to process the differential file against the master file for this assignment. In real life situations, you should then process the differential file against the master file in batch but this is not needed for this assignment.

Part C: On line processing with hashing

For this part, you should read the given master file and create a new version of the master file where the position of a record is determined by hashing. Then, you have to process the (unordered) transaction file against the hashed master file.
You can choose the hashing function and the collision resolution that you prefer. Of course, you must choose the size of the new master file in such a way that collision exist (part of the marks will be given for your collision resolution approach). Finally, do not sort the transaction file; instead, process them sequentially.

Your program

For this assignment, your program processes a master file called **master** (ordered by identifier), and a transaction file called **trans** (unordered) with the same description as in assignment 1.
In part A, your program should externally sort **trans** creating a file called **newtrans**. Call your executable program **extsort**. We will use the following command to run your program:

```
extsort trans newtrans
```

In part B, your program should process the transactions against the **master** file, creating and using a differential file called **diff**. Call your executable program **binary**. We will use the following command to run your program:

```
binary master trans diff
```

In part C, you have to give two programs. The first one should create the **newmaster** file using hashing. The second one should process the transactions against the **newmaster** file. Call your executable programs **hashmast** and **hashproc**, respectively. We will use the following commands to run your programs:

```
hashmast master newmaster
hashproc newmaster trans
```

Your programs **binary** and **hashproc** must give the average time for each kind of operation (insertion, modification and deletion). [Hint: use `gettimeofday`.]
Do not read the entire transaction file 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. Your program can consider only errors that depend on the present content of the master file (for instance, a change in a nonexisting flight).

How to hand in your assignment

First, you must submit your files. This involves four programs: `extsort.c`, `binary.c`, `hashmast.c`, `hashproc.c`, and the error output `error.log`. Use the submit command to submit copies of your source codes:

```
submit -N a2 csc228h extsort.c binary.c hashmast.c hashproc.c error.log
```

At the beginning of the tutorial of November 4, you must hand in source listing for your C programs (Unix) and your output data files. This includes your input files (**master** and **trans**), and your output files (**newtrans**, **master**, **newmaster** after hashmast, and **newmaster** after hashproc). 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, commented tests in the output files, and so on. The marker must be able to compile your program using **gcc**, and run it successfully on an appropriate test file.