CSI 2131 File Management — Winter 2002
Computer Science — University of Ottawa

Homework Assignment #3 (100 points, weight 10%)
Due: Tuesday, March 26, 2:00 p.m., at the box CSI2131A or CSI2131B (MCD, 3rd floor)

---

1 Problem 1: Programming Assignment (80 marks)

In this assignment, you are going to create a hashed index for a datafile of the University of Ottawa courses currently offered (during the Winter 2002 section).
The datafile is organized as a variable-length record file with fields separated by commas ",". The datafile has the following fields:

- course code (primary key) in format: AAA9999
- course name
- faculty
- department (sometimes absent)

Your programming assignment consists of two parts. In Program A, you will be constructing the hashed index file and computing some statistics about it. In program B, you will be searching the datafile for information about courses, using the hashed index file constructed in Program A.

Important note on design and implementation:

Both Program A and Program B use a class called "HashFile".
First, implement HashFile for Program A, test it and save it in files `hashfileA.cpp` and `hashfileA.h`.
Second, implement HashFile for Program B, storing it in files `hashfileB.cpp` and `hashfileB.h`.
You may add methods to the class implemented for Program A or modify it creating a different class.
At the end, the class in `hashfileA.*` and `hashfileB.*` may be two different classes (one specifically for Program A and one for Program B) or they may be exactly the same class.
In any case, **please submit them as separate files:** `hashfileA.*` and `hashfileB.*`. The first one will be used for compiling and running program A and the second one for program B.
In this way, we are allowing for the elegance of a unique class, but as well for independent testing and progressive implementation (one class may be working, while the other may still have errors at the due date time).

1.1 Program A: Building a Hashed Primary Index (60 marks)

The first program that you will write must build **a hashed primary index file** with the following characteristics:

- fields for records:
  - course code (7 characters)
  - byte offset for course record in datafile (7 characters)
- records terminated by 2 special characters `\r\n` (for legibility under DOS); note that for text files you write `"\n"` in order to write both characters.
- file characteristics: fixed-length, hashed, text file
- hashing organization: hashing with 1 record per address and with collision resolution by progressive overflow.
- hash function: returns the sum of the ASCII codes for the characters in the course name **mod** the number of hash addresses.

Note: It would be much more space efficient to store the byte offset as a number in binary notation (using 3 bytes), rather than using 7 characters (7 bytes); we decided to use characters to simplify the assignment and to make the file readable, allowing you and the markers to examine its contents.

Specifications for Program A:

- Input data: datafile, index file name, number of hash addresses.
- Result: Create a hashed index file for the given datafile following the specifications above. Calculate the average search length and other statistics.
- Output inside the program: Print all input data values (input and output file names and number of hash addresses) as well as the following statistics on the hashed index file created:
  - the average search length (sum of the search length for each key divided by the number of keys);
  - the number of records at their home address and the number of records not at their home address;
  - the packing density;
  - the size of the hashed index file (in bytes) (by checking the offset at the end of the file)- for marking verification only.
- Program files to be handed in for Program A:
  - hashbuild.cpp (main program)

  - hashfileA.cpp, hashfileA.h: class HashFile for creating and updating the hashed file. Useful suggested methods: create (empty) hashed addresses, insert key, hash function, positioning at address i, read/write record, etc.

- Printouts to be handed in for Program A:
  - printouts for all program files handed in
  - printouts for the outputs for all tests requested (available from the web page)
  - printout for the hashed index file only for the tests for which this is specifically requested (specified in the web page).

Note: Before inserting keys at the hashed index file, you need to create the "empty addresses" in the file. This can be accomplished by printing blank records (14 blank characters plus the two end-of-line characters for each hash address).

1.2 Part B: Searching the datafile using the primary index (20 marks)

The second program that you will write must search the datafile using the **hashed primary key index** that you have built in part A.
Your main program must do the following tasks in this order:

- Read the input data (datafile name and hashed indexfile name).
- Test the search method (in HashFile class) by searching for "CSI2131". This search method (method in class HashFile) must print all the data for a course in a format easily readable by a human being (your choice); print also the search length or number of "probes" used in the search.
- Ask the user for 'Y' or 'N' answer to question "Would you like to search?". If answer is 'N', terminate the program. If the answer is 'Y', ask the user for a course number to search, and perform the search.

Specifications for Program B:

- Input data: datafile name, index file name.
- Output inside the program: output all input data values, and the following info:
  - output generated by the search for CSI2131
  - output generated by the extra search, if applicable.
- Program files to be handed in for Program B:
  - hashsearch.cpp (main program for Part B)
  - hashfileB.cpp, hashfileB.h: class HashFile for searching on a previously created hashed file. Useful suggested methods: open existing hashed file (loading relevant info), search, hash function, positioning at address i, read record, etc.

- Printouts to be handed in for Program B:
  - printouts for all program files handed in
  - printouts for the outputs for all tests requested (available from the web page)

1.3 Bonus marks (up to 10 marks)

We advise you to work on bonus marks only after the basic part of the assignment has been completed. Save the previous version of your assignment before making bonus additions to be safe that you have a working basic program at the due date. To be eligible for bonus marks you must have implemented all the features of the basic parts for Program A and Program B.

Up to 10 bonus marks will be given if you make your problem more general, using hashing with buckets containing more than one record per address. In this case, bucket size (number of records per address) must be an input value entered by the user. In this case, you must read one bucket at a time, rather than one record at a time. Also a "probe" corresponds to reading one bucket (rather than one record); this would affect the average search length. If a bucket of size 1 is specified, the program should behave like the standard case without buckets.

When implementing this, you would have to make several decisions about your file structures and algorithms. These decisions are part of your bonus marks, so you are not supposed to be asking a lot of questions from us on how to overcome every single difficulty.

In order to have your bonus marks considered, you must include a page (the first page before the printout of your program code). The page must be entitled "Entry for bonus marks". There, you should explain to the marker how buckets have been implemented and any implementation decisions you have made. Be clear and concise, but do emphasize the good aspects of your implementation, since this will guide the TA to decide how many marks to award you out of the 10 bonus marks.

2 Problem 2: Written Assignment (20 marks)

2.1 External Sorting (10 marks)

Consider the algorithm for merging as a way of sorting large files on disk of Section 8.5 of the textbook, and assume that you are given a file with the following description:

```
File Description
----------------
   Number of Records = 500,000
   Record Size = 200 bytes
```

and a system with the following description (In addition, memory is available for holding the program, the operating system, and the necessary low-level data structures, variables, etc. used by the program):

```
System Description
------------------
   Memory available as work area = 1,000 KBytes
   Number of Output Buffers = 2
   Size of the Output Buffers = 100 KBytes/each
   Seek time + Rotational Delay = 10 msec/block
   Transmission time = 10,000 bytes/msec
```

As described in Section 8.5, the algorithm is divided into 4 steps:

Step 1: Reading Records into Memory for Sorting and Forming Runs

Step 2: Writing Sorted Runs to Disk

Step 3: Reading Sorted Runs into Memory for Merging

Step 4: Writing Sorted File to Disk

Please answer the following questions:

1. In how many runs does the data file need to be divided in order to run this program?
2. How many records from each run will be held in memory simultaneously during Step 3?
3. How many seeks are necessary for each of the four steps listed above (indicate the number of seeks for each step separately)?
4. How much time (taking seek time, rotational delay and transmission time into consideration) is necessary for the overall algorithm on this file?

2.2 Hashing (theory) (10 marks)

We assume that $1,000$ addresses are allocated for the storage of 600 records in a randomly hashed file.

1. What is the *packing density* of this file?
2. What is the expected number of addresses having *no* record assigned to them?
3. What is the expected number of addresses having *exactly one* record assigned to them?
4. If we assume that only a single record can be affected to each home address, how many *overflowing records* can be expected?
5. What percentage of records are *overflowing records*?
6. What is the *packing density* of this file if buckets of size 2 are used?
7. What percentage of records are *overflowing records* when a bucket size of 2 is used?

Standards

You must hand in, on a labeled large envelope:

- stapled sheets of paper of your handwritten or typed answer for Problem 2;
- printouts to be handed in for Program A and Program B as specified before.
- a 3.5-inch DOS/Windows diskette containing only: `(*.h, *.cpp)` for Program A and Program B; all files must have the names specified above and must be at the root directory (no subdirectories, please).