CSI 2131 File Management — Winter 2004
Computer Science — University of Ottawa

Homework Assignment #2 (100 points, weight 10%)

Due: Thursday, March 4th at 5:00 p.m. (via webCT)

Simple Indexing

1 Problem description

You will write a C++ program that creates an index for a datafile, and maintains the indexed datafile, allowing various operations such as search, addition, deletion and updates.

Datafile description:
The datafile consists of course records in a similar format as in assignment#1 output file. The only difference is that the file is not sorted by course code.

Index file description:
The index file starts with a flag (1 byte) that is on if the index is not up-to-date; you may decide how to encode `on` and `off`. Next , it contains a number (2 bytes) which indicates the number of records in the index file. This is followed by the index records. Each record is of fixed length (11 bytes): the first 7 bytes store the key (course code) and the next 4 bytes store the reference field (byte offset of the corresponding record in the datafile).

Creating an index and maintaining an indexed datafile
You will write a class `indexedFile` that will maintain the datafile and a simple index that is fully held in main memory. The index is kept in an array sorted by the primary key (course code). After usage, the index should be saved into a file (to be persistent), and before usage the index file should be loaded to main memory, but most operations on the index will be done in main memory.
Your class `indexedFile` will provide public methods to handle the following operations:

- Initialization:
  The class constructor will receive a file name which will be the name of the datafile to be used. Any other initialization necessary is done by the constructor.

- Open indexed datafile: method `open`
  This operation should open the datafile and load the index. To load the index, it will check whether the index file exists. The index file will have the same name as the datafile but with the prefix `index` before it. If it exists, the method will check if the index is up-to-date (check a flag at the beginning of the index file), and if so it will load it into an array in main memory. If the index file does not exist or is not up-to-date, it will build the index in main memory by going through the records in the datafile.

Note that you are responsible for setting the flag at the beginning of the index file, whenever it becomes not up-to-date in any of the following operations below.

- Close indexed datafile: method `close`
  This operation will close the datafile. It will also save and close the index file. If the index file doesn't exist or if it exists but it is not up-to-date, this method will write the index file, using the information on the array in main memory.

- Search in index (private method): method `indexSearch`
  This operation will receive a course code and it will search for the byte offset of the course's data record, by using the index in main memory. You should use binary searching on the index array.

- Search for course: method `recordSearch`
  This operation will receive a course code and call the method `indexSearch`, and if found, it will read and return the data record.

- Add record: method `addRecord`
  This operation will receive a new record to be added to the datafile. The index in main memory should be updated accordingly. Note that you must make sure that records for existing courses will not be added again.

- Delete record: method `deleteRecord`
  This operation will receive the course code for the record to be deleted. If the record exists, the deletion will be done by "marking" the record as deleted with an `*` on top of its first character after the length indicator. The corresponding entry should also be removed from the index in main memory.

- Storage compaction: method `storageCompaction`
  This operation is going to remove the deleted records from the datafile. The best approach is to do the following steps: create a new file without the deleted records, close both original and new file, remove original file, rename the new file as the original file, reopen the file. The index in main memory must be updated as you create the new file. Note: for removing and renaming the files, you can use the following functions: `int remove(char* filename)`, `int rename(char* oldname,char* newname)`.
  To use these functions the files must be closed.

- Update record: method `updateRecord`
  This operation will receive a course code and a new record with updated information. If the course code and record size are unchanged, this operation will only change the datafile. If the course code or the record size changes, this operation will perform a deletion and addition.

- Simulated power off: method `powerOff`
  This operation is going to simulate an interruption of the correct termination of the

program. In this operation, you simply close the open files, but don't do the proper updates on the index file. This is going to be used to check whether you reconstruct the index properly, next time we use the operation "open indexed file".

Most methods will return true or false depending whether the operation was successful. Important: remember that whenever you position at the of end-of-file, the "fail" flag will be on, and won't allow you to do anything else on the file after that. In order to proceed, you need to use the method `clear()` which will clear all the status flags for the file.

2 Implementation details and standards

Don't forget: your program will be compiled and run with the dotnet compiler; so it must successfully compile and run when using this compiler.

Program structure:
We will provide you with the following files which should be the files in your project:

- `main.cpp`: Tester program that will use the classes you will create. This tester program should not be altered. Note that we may create some other variants of the tester program that might test some other aspects of your classes. For this reason it is essential that you do not modify class names, method names or parameter types.
- `record.h`: The header file with the `record` class definition. You may add elements to this class definition, but you should not alter the definitions for the methods included there.
- `record.cpp`: A file with dummy implementations of the methods for the the `record` class. To be completed by you.
- `indexedfile.h`: The header file with the `indexedFile` class definition. You may add elements to this class definition, but you should not alter the definitions for the methods included there.
- `indexedfile.cpp`: A file with dummy implementations of the methods for the the `indexedFile` class. To be completed by you.

Documentation and Style
Please follow the same guidelines as in assignment#1.

What and how to submit your assignment
Create a directory/folder named `a2` containing only the following files: `record.cpp`, `record.h`, `indexedfile.cpp`, `indexedfile.h`. EVERY FILE MUST CONTAIN A HEADER WITH Student Name, Student Number, Course and Section. Zip the folder and submit the zipped file as your assignment#2 submission to webCT (only one file is submitted).