CS3114 (Fall 2009)

Project #2

Due Wednesday, Oct 28 @ 11:55 PM for 100 points
Initial Schedule due Wednesday, October 14 @ 11:55 PM
Revised Schedule due Wednesday, October 21 @ 11:55 PM
Revised 10/6/2009

Project Description: For this project, you will implement an external sorting algorithm for binary data. The input data file will consist of $8 \times N$ blocks of data, where a block is 4096 bytes. Each block will contain a series of records, where each record has 8 bytes. The first 4-byte field is an **unsigned integer** value for the record id and the second 4-byte field is a **float** value for the key, which will be used for sorting. Thus each block contains 512 records.

Your job is to sort the file (in ascending order of the key values), as follows. Using replacement selection (as described in the class notes), you will sort sections of the file in a working memory that is 8 blocks long. To be precise, the heap will be 8 blocks in size; in addition you will also have a one block input buffer, a one block output buffer and any additional working variables that you need. To process, read the first 8 blocks of the input file into memory and use replacement selection to create the longest possible run. As it is being created, the run is output to the one block output buffer. Whenever this output buffer becomes full, it is written to an output file called the *run file*. When the first run is complete, continue on to the next section of the input file, adding the second run to the end of the run file. When the process of creating runs is complete, the run file will contain some number of runs, each run being at least 8 blocks long, with the data sorted within each run. For convenience, you will probably want to begin each run in a new block.

You will then use a multi-way merge to combine the runs into a single sorted file. **You must reuse the 8 blocks of memory used for the heap in the run-building step to store working data from the runs during the merge step.** Multi-way merging is done by reading the first block from each of the runs currently being merged into your working area, and merging these runs into the one block output buffer. When the output buffer fills up, it is written to the another output file. Whenever one of the input blocks is exhausted, read in the next block for that particular run. This

step requires random access (using *seek*) to the run file, and a sequential write of the output file. **Depending on the size of all records, you may need multiple passes of multiway-merging to sort the whole record.**

Invocation and I/O Files:

The program will be invoked from the command-line as:

```
extsort  <data-file-name>  <stat-file-name>
```

The data file `<data-file-name>` is the file to be sorted. At the end of your program, the data file (on disk) should be sorted. So this program does modify the input data file. Be careful to keep a copy of the original when you do your testing.

In addition to sorting the data file, you must report some information about the execution of your program.

1. You will need to report part of the sorted data file to standard output. Specifically, your program will print the first record from each 4096-byte block, in order, from the sorted data file. The records are to be printed 5 records to a line (showing both the key value and the id value for each record), the values separated by whitespace and formatted into columns. This program output must appear EXACTLY as described; ANY deviation from this requirement will result in a significant deduction in points.

2. You will generate and output some statistics about the execution of your program. Formatting does not matter as long as we can tell what each statistic is. Write these statistics to `<stat-file-name>`. Your code will be tested with different sample files. Make sure your program DOES NOT overwrite `<stat-file-name>` each time it is run; instead, have it append new statistics to the end of the file. The information to write is as follows.

   (a) The name of the data file being sorted.

   (b) The time that your program took to execute the heapsort. Put two calls to the standard Java timing method "`System.currentTimeMillis()`'" in your program, one at the beginning and another at the end. This method returns a long value. The difference between the two values will be the total runtime in milliseconds. You should ONLY time the sort, and not the program output as described above.

Programming Standards:

You must conform to good programming/documentation standards. Some specifics:

- You must include a header comment, preceding main(), specifying the compiler and operating system used and the date completed.
- Your header comment must describe what your program does; don't just plagiarize language from this spec.
- You must include a comment explaining the purpose of every variable or named constant you use in your program.
- You must use meaningful identifier names that suggest the meaning or purpose of the constant, variable, function, etc.
- Always use named constants or enumerated types instead of literal constants in the code.
- Precede every major block of your code with a comment explaining its purpose. You don't have to describe how it works unless you do something so sneaky it deserves special recognition.
- You must use indentation and blank lines to make control structures more readable.
- Precede each function and/or class method with a header comment describing what the function does, the logical significance of each parameter (if any), and pre- and post-conditions.
- Decompose your design logically, identifying which components should be objects and what operations should be encapsulated for each.

Neither the GTAs nor the instructors will help any student debug an implementation unless it is properly documented and exhibits good programming style. Be sure to begin your internal documentation right from the start.

You may only use code you have written, either specifically for this project or for earlier programs, or code taken from the textbook. Note that the textbook code is not designed for the specific purpose of this assignment,

and is therefore likely to require modification. It may, however, provide a useful starting point.

Testing:

A sample data file will be posted to the website to help you test your program. This is not the data file that will be used in grading your program. The test data provided to you will attempt to exercise the various syntactic elements of the command specifications. It makes no effort to be comprehensive in terms of testing the data structures required by the program. Thus, while the test data provided should be useful, you should also do testing on your own test data to ensure that your program works correctly.

Deliverables:

When structuring the source files of your project, use a flat directory structure; that is, your source files will all be contained in the project root. Any subdirectories in the project will be ignored. If you used a makefile to compile your code, be sure to include any necessary files or instructions so that the TAs can compile it.

Your submission should be archived as one file, which should contain all of the source code for the project, along with any files or instructions necessary to compile the code. If you need to explain any pertinent information to aid the TA in the grading of your project, you may include an optional "README" file in your submitted archive.

You will submit your project through the scholar/??? system. If you make multiple submissions, only your last submission will be evaluated.

Scheduling:

In addition to the project submission, you are also required to submit an initial project schedule and a revised schedule before the corresponding deadlines. You won't receive direct credit for submitting the schedules as required, but each instance of failing to submit scheduling information as required will lose 5 points from the project grade.

Pledge:

Your project submission must include a statement, pledging your conformance to the Honor Code requirements for this course. Specifically, you must include the following pledge statement near the beginning of the file containing the function main() in your program. The text of the pledge will also be posted online.

```
//  On my honor:
//
//  - I have not used source code obtained from another student,
//    or any other unauthorized source, either modified or unmodified.
//
//  - All source code and documentation used in my program
//    is either my original work, or was derived by me from the source
//    code published in the textbook for this course.
//
//  - I have not discussed coding details about this project with anyone
//    other than my instructor, ACM/UPE tutors or the GTAs assigned to this
//    course. I understand that I may discuss the concepts of this program
//    with other students, and that another student may help me debug my
//    program so long as neither of us writes anything during the discussion
//    or modifies any computer file during the discussion.  I have violated
//    neither the spirit nor letter of this restriction.
//
```

Programs that do not contain this pledge will not be graded.