CS2604 (Fall 2000)
PROGRAMMING ASSIGNMENT #3
Due Wednesday, October 25 @ 11:00 PM for 100 points
Early bonus date: Tuesday, October 24 @ 11:00 PM for 10 point bonus
Late penalty: 10 points/day with final due date Saturday, October 28 at 11:00 PM

Assignment:
You will implement an external sorting algorithm for binary data. The input data file will consist of many 4-byte records, with each record consisting of two 2-byte (short) integer value in the range 1 to 30,000. The first 2-byte field is the key value (used for sorting) and the second 2-byte field contains a data value. The input file is guaranteed to be a multiple of 4096 bytes. All I/O operations will be done on blocks of size 4096 bytes (i.e., 1024 logical records).

Warning: The data file is a binary file. For information on processing binary files in C++, see URL http://courses.cs.vt.edu/~cs2604/fall00/binio.html.

Your job is to sort the file (in ascending order), using a modified version of the Heapsort. The modification comes in the interaction between the Heapsort algorithm and the file storing the data. The heap array will be the file itself, rather than an array stored in memory. All accesses to the file will be mediated by a buffer pool. The buffer pool will store 4096-byte blocks (1024 records). The buffer pool will be organized using the Least Recently Used (LRU) replacement scheme. See Section 9.3 in the book for more information about buffer pools.

Design Considerations:
The primary design concern for this project will be the interaction between the logical heap as viewed by the Heapsort algorithm, and the physical representation of the heap as implemented by the disk file mediated by the buffer pool. You should pay careful attention to the interface that you design for the buffer pool, since you will be using this again in Project 4. In essence, the disk file will be the heap array, and all accesses to the heap from the Heapsort algorithm will be in the form of requests to the buffer pool for specific blocks of the file.

Invocation and I/O Files:
The program will be invoked from the command-line as:

`heapsort <data-file-name> <numb-buffers>`

The data file `<data-file-name>` is the file to be sorted. The sorting takes place in that file, so this program does modify the input data file. Be careful to keep a copy of the original when you do your testing. The parameter `<numb-buffers>` determines the number of buffers allocated for the buffer pool. This value will be in the range 1–20.

At the end of your program, the data file (on disk) should be sorted. Do not forget to flush buffers from your bufferpool as necessary.

In addition to sorting the data file, you must report some information to the standard output stream. This output for your program must appear EXACTLY as follows. ANY deviation from this requirement will result in a significant deduction in points. The information printed will consist of two parts.

The first part will consist of the first record from each 4096 byte block, in order, from the final sorted output. The records are to be printed 8 records to a line (showing both the key value and
the data value for each record) with the values separated by whitespace, formatted so that they line up in columns.

The second part will be the time that your program took to execute. Put calls to “\texttt{clock()}” in your program, one at the beginning and another at the end. This function is available by using \texttt{#include <time.h>}.” It returns a \texttt{clock_t} result that is compatible with long integers. The difference between the two values will be the total time in “clock ticks.” Divide this number by \texttt{CLOCKS_PER_SEC} to get total time in seconds. An example will be posted to the website.

\textbf{Programming Standards:}

You are expected to observe good programming/documentation standards, as described in the Elements of Programming Style. 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. You may not use code from STL, MFC, or a similar library in your program.

\textbf{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. 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:
You will submit this project electronically. In particular, you will create a zip'ed archive file containing the following items (and nothing else):

- all source code files necessary to build an executable
- either the project workspace files (.dsw and .dsp) for Visual C++ users, or a makefile for g++ users.
- A document in either plain ASCII text, Postscript, PDF, or MS Word format, that documents how your program is designed and why you made those design choices. The document should discuss at least, but need not be limited to, the design issues described in the “design considerations” section above.

Don’t forget that the name of your executable should be heapsort. Windows users should be sure to use a modern zip tool which preserves long file names. A suitable freeware command-line zip tool will be posted on the course website. UNIX users should submit a gzip’ed and tar’ed file.

Once you have assembled the archive file for submission, for your own protection, please move it to a location other than your development directory, unzip the contents, build an executable, and test that executable on at least one input file. Failure to do this may result in delayed evaluation of your program, and a loss of points.

You will submit your project to the automated Curator server as “Project 2”. The instructions and necessary software are available at: http://ei.cs.vt.edu/~eags/CuratorGuides.html. If you make multiple submissions, only your last submission will be evaluated.

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 in the header comment preceding the function main() in your program. The text of the pledge will also be posted online.

// On my honor:
//
// - I have not used C++ language code obtained from another student,
//   or any other unauthorized source, either modified or unmodified.
//
// - All C++ language 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.