CSI2131, Winter 2001
Assignment 3
Due on Monday, March 26, 11:00am

1 Written Problem : External Sorting

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
----------------
   File Size =   50,000 KBytes
   Number of Records = 500,000
   Record Size = 100 bytes
```

and a system with the following description:

```
System Description
------------------
   Memory available as work area = 500 KBytes
   Number of Output Buffers = 2
   Size of the Output Buffers = 100 KBytes/each
```

(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)
As described in Section 8.5, the I/O operations can be 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 for using this algorithm?
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 seek for each step separately.

2 Programming Problem: Co-Sequential Processing for Merging Large Files

A large first year CSI course is divided into 4 sections, A, B, C and D of 100 students each. Each section has its own database recording the student number, the expected year of graduation, and the grade obtained in the course. The lists are ordered by student number. The professor would like to

- Merge the four sections into a single database also ordered by student number.
- Compute the average grade per year of graduation (irrespective of the student's section).

You are required to provide the software necessary for merging the lists and computing these statistics, noting that these two tasks have to be performed in the course of a single co-sequential process. In other words, the statistics should be gathered **during** the merging process.

In more detail, you are asked to implement the K-way Merge Algorithm described in Section 8.3.1 of the Textbook. Despite the fact that only 4 lists of 100 students are given here, your algorithm should work for any number K of lists of any size (each list does not necessarily have the same size).

The student records have the following structure:

Field Name	Type and length of field	Note
Student Number	7 characters (numeric)	Unique Key
Year of graduation	2 characters (numeric)	
Final Grade	3 characters (numeric)	

The fields within a record are not separated, but each record is separated by a newline character.

We assume that the system description is the following:

```
System Description
------------------
   Number of Student Records that can fit
   simultaneously in the work area         = 40 records

   Size of the Output Buffer (we use
   only one in this application)           = 60 records
```

Given the system's description, it is clear that the lists cannot fit into memory and must be kept in secondary storage (Note: This numbers are obviously not realistic; the numbers were scaled down for simplicity. The same method can be used for merging files with sizes exceeding more realistic figures of available memory!).

Specific requirements are listed below:

- **Input Buffers**
  As in Step 3 of Problem 1, you are required to read as many records as possible at a time (in the case of the files provided, your program will read 10 records from each list, so that at any given time the program is holding 40 records in main memory). That is, your program must include buffers containing 10 records per input file.

- **Output Buffer**
  As in Step 4 of Problem 1, your program will manage an output buffer (60 records in this case), so that records are only written to the file when the buffer is full or when the cosequential processing is finalized.

- **Cosequential processing and merging algorithm made general**
  Try to make your co-sequential processing as general as possible. Your K-way merge should be general for any number of lists or any sizes of lists (the sizes don't have to be known a priori). The size of the work area (40) and the size of the output buffer (60) can be constants defined in your program (easily modifiable if these numbers were to change).
  But not so general ...
  You are not required to be as general as the book is, so that it is OK if your cosequential processing only works for the given type of records. Your program may also have a constant (set before the program reads the files) that stores the smallest graduating year in the files (2001 for the given files); this may facilitate your statistics gathering.

- **Design issues**
  The gathering of statistics can be done when an item is "processed" by the co-sequential processing. Buffering and statistics gathering can be elegantly hiden from the main merging algorithm. Buffering can be handled by two classes (one for input buffering and one for output buffering). Statistics gathering can be handled by a method responsible for processing an item.

- **Resulting merged file, output report and standards**
  Your program should write the merged list to a file named `mergedlist.txt` and write a report of statistics to a file named `statistics.txt` containing the following information:

  - the first 15 records of mergedlist.txt;
  - the statistics themselves:
    * The average grade of students graduating in year 2001 is:
    * The average grade of students graduating in year 2002 is:
    * etc...

  The file `statistics.txt` should be printed and handed in together with your listings.

  Do not prompt the user for file names. Your "main" program can open the 4 input files and the output file; the "merge" method can receive the logical file names from the main program (i.e. an array of input files, the number of input files and the output file).

Further details about the format of your program are included in the standards published in the web page. File names for the various parts of the program are specified there.

- **Documentation**
  Write lots of comments in your code explaining what you are doing. Since you have some relative freedom in how to design your program and various classes, you should write some introductory comments in your program, explaining what is being done and where. Right at the beginning of the main program file, please, include a note on the status of your program: if it compiles, if it runs successfully, which features were implemented and which ones were not implemented, known bugs or cases the program does not handle, if any. There will be discounts, of course, if the parts of the program don't work, but declaired errors will be looked at with more sympathy by the TAs; this will greatly facilitate marking!