The University of Chicago
Department of Computer Science

CMSC 15200 – Introduction to Computer Science 2
Summer Quarter 2005
Homework #4 (08/05/2005)
Due: 08/10/2005

Name:

Student ID: *Instructor:* Borja Sotomayor

Do not write in this area

1	2	3	4	5	**TOTAL**

Maximum possible points: 75

Exercise 1 <<15 points>>

Modify exercise 2 from homework #2 so that the user will be able to place bets in each game:

- The user starts with $100.
- In the guessing game, the user will place a bet and the prize will be proportional to the bet:
  - Guess on the first try: 40 x bet
  - Guess on the second try: 20 x bet
  - Guess on the third try: 5 x bet
  - Guess on the fourth try: 2 x bet
  - Guess on the fifth: 1 x bet (the user gets his/her money back)
- Playing the lottery always costs $1, and has the following prizes:
  - Guessing all six number: $1,000,000
  - Guessing any five numbers: $100,000
  - Guessing any four numbers: $100
  - Guessing any three numbers: $10
  - Guessing any two numbers: $1 (the user gets his/her money back)
- Users can play the game as many times as they want, until they run out of money.

Exercise 2 <<15 points>>

Modify the hangman game exercise 4 from homework #2 (the hangman game) so that the program chooses a word at random from a text file with one word per line. You do

not know the size of the file in advance, so choosing a word at random will not be as simple as choosing a number between 1 and N. If you get stumped on how to do this, simply read the first word in the file.

Note: The 15 points in this exercise are divided this way:

- Five points for reading a word from the file (random or not)
- Five points for reading a *random* word from the file.
- Five points if you don't hardcode the filename in your code. In other words, instead of limiting your program to always using a word file with the same name, the user should be able to run the program like this:

```
$ ./hangman <wordfile>
```

Note (2): A sample file called `hangmanwords` is provided in the homework files (available at the course website)

Exercise 3 <<20 points>>

You will implement a stripped-down version of the classic UNIX command `cat`. This command is generally run like this:

```
cat <file1> <file2> <file3> ... <fileN>
```

Each command-line argument refers to a text file on the user's filesystem. The program reads every file and outputs the con**cat**enation of all the files. Note that if you specify a single file, then the cat command will simply output the contents of that single file.

You must make sure you capture any errors that might occur when running the command. For example, if the user specifies an invalid filename, you must say so.

For full credit (15 points otherwise), you will assume that the last file specified is an *output* file where the concatenation of all the previous files must be written to. For example, if the user runs the command like so:

```
cat foo.txt bar.txt baz.txt
```

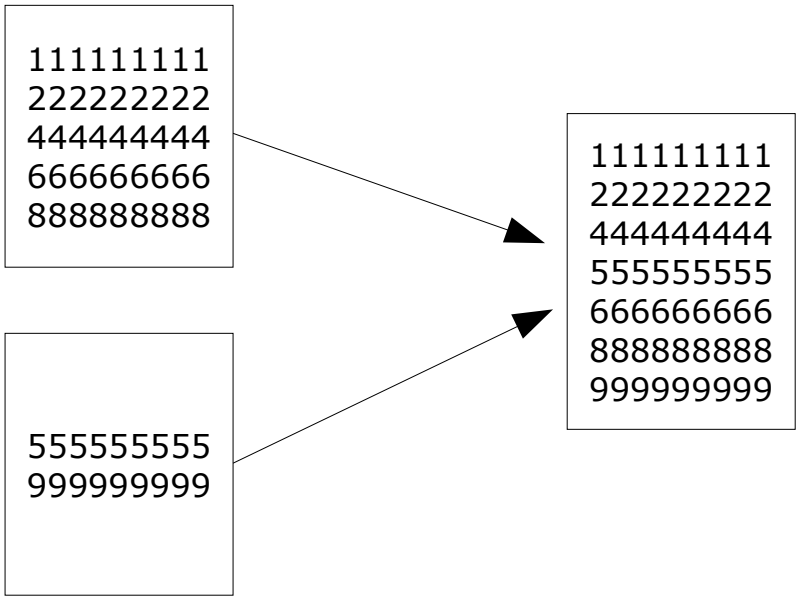
Then the program will concatenate foo.txt and bar.txt and store the result in baz.txt. If the user specifies a single file, then the program will still behave as described above.

Hint: To do this exercise you do *not* need to load the entire files into memory.

Exercise 4 <<20 points>>

PhoneCorp and PhoneTech, the two biggest phone companies in the US, have just completed a corporate merger. They are now faced with the daunting task of *merging* their client data files into a single file. In particular, each company has a text file with the social security numbers of their clients (one 9-digit number in each line) in increasing order. Your task is to create a program that takes those two files and creates a new file with the numbers from both files, in increasing order. You can assume that the two file have no numbers in common (i.e. the two sets of clients are disjoint)

For example:



Your program must be run like this:

```
mergefiles <clientfile1> <clientfile2> <result>
```

For full credit (10 points otherwise), your file must perform the merge doing a *single pass* through each of the files, without loading them into memory.

Hint: You are asked to perform a *merge* of two sequences of data (in this case, integers). This is a very common task in programming, and there is a very well known algorithm that does this (meeting the requirements for full credit). You are encouraged to look it up on books/websites/etc. as long as you cite your sources.

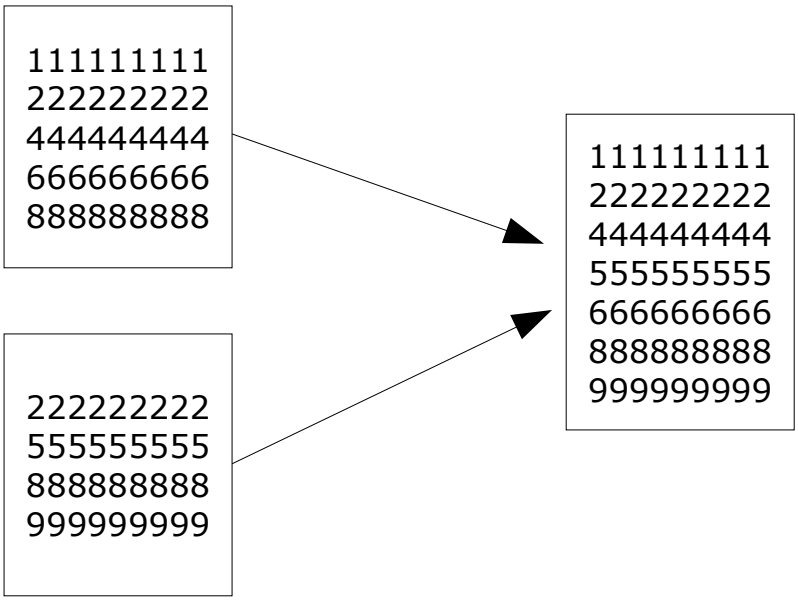
Note: Two example files (`clients1` and `clients2`) are provided in the homework files. The result of correctly merging these two files is the following:

```
113624982
135648623
154564564
168943568
193215689
236584654
276546846
279461322
283216569
295464654
316243213
333218954
342654658
345654858
393546562
395221354
399321545
412135468
422354943
462213589
493215812
539565482
539635482
783213542
896546543
973213215
993219953
```

Hint 2: This is a good example of an exercise you should try to first solve with a more reduced problem set, before approaching the complete problem. For example, to familiarize yourself with the merging algorithm (without dealing with all the I/O messiness), try merging two 5-position arrays (preloaded with any integers you want, as long as they are in increasing order) into a 10-position array. When you do start to add the I/O code, first give your algorithm a try with smaller files than the ones provided (with single-digit integers, for example, which are easier to check than 9-digit numbers).

Exercise 5 <<5 points>>

Make a simple modification to Exercise 4 so that you program will be able to handle files with common numbers (i.e. the two companies share some clients in common, so the two sets of clients are *not* disjoint). For example:



Note: Two example files (`clients1_rep` and `clients2_rep`) are provided in the homework files. The result of correctly merging these two files is the following:

```
113624982
135648623
168943568
236584654
256684623
276546846
279461322
295464654
316243213
333218954
342654658
345654858
356698823
393546562
395221354
399321545
399645652
412135468
419654332
422354943
462213589
539565482
539635482
712315465
896546543
936532132
946546523
982132132
983213223
```