CSE 303, Spring 2009
Midterm Exam
Wednesday, May 6, 2009

Personal Information:

Name: _____

Student ID #: _____

- You have 50 minutes to complete this exam.
  You may receive a deduction if you keep working after the instructor calls for papers.
- This exam is open-book/notes. You may not use any computing devices including calculators.
- Code will be graded on proper behavior/output and not on style, unless otherwise indicated.
- Do not abbreviate code, such as "ditto" marks or dot-dot-dot ... marks.
- If you enter the room, you must turn in an exam before leaving the room.
- You must show your Student ID to a TA or instructor for your exam to be accepted.

Good luck!

Score summary: (for grader only)

Problem	Description	Earned	Max
1	Unix/Linux Shell		20
2	Regular Expressions		20
3	Bash Shell Scripting		20
4	C Pointers		20
5	C Programming		20
TOTAL	**Total Points**		**100**

1. Unix/Linux Shell

Describe in English what the following command does. Be specific.

a) `ls *.sh *303* | uniq > myfiles.txt`

Write a shell command to perform each of the following actions. For full credit, each must be a single command line without any use of the semicolon `;` or `&&` or `||` operators. You may use redirection such as `<`, `>`, and `|`.

b) Give read permission (only) for all users to all `.doc` files in user `oterod`'s home directory. (Your command should work regardless of the current directory at the moment it is typed.)

c) Output the number of files in the current directory whose names contains the current user's username. For example, if the current username is `joe`, the file `sloppyjoerecipe.txt` would be among those counted.

d) Output the names of the first 3 files in the current directory whose text contains the word `password`, in reverse alphabetical order. If there are not 3 such files, output however many there are.

2. Regular Expressions

Write a regular expression that would match each of the following patterns, using standard "extended" syntax as shown in class. You are not writing a complete Unix command here, just a regular expression.

a) Match all lines that begin with the letters `CSE` and also contain the letters `CSE` again later in the same line.

b) Match all lines that contain the same CSE course number twice. A CSE course number begins with the string `CSE`, followed by an optional space, followed by a 3-digit integer. For example, `CSE 303` and `CSE190` are valid course numbers. You are to match lines that contain the same course number twice, such as the line:

```
I am in CSE 303 ...  and I think CSE303 is a hard class!
```

c) **Write a complete shell command** that will examine a file named `notes.txt` and look for lines that contain the same CSE course number twice, as described in (b). Every such entire line should be replaced with `CENSORED` followed by the course number. For example, the entire line in part (b) would be replaced by:

```
CENSORED 303
```

Your command should output the contents of `notes.txt` to the console with the preceding changes made to it. You are not actually modifying the contents of `notes.txt`, merely outputting to the console. Your answer must be a single command line without any use of the semicolon `;` or `&&` or `||` operators. You should use exactly 1 regular expression-related command in your solution.

3. Bash Shell Scripting

Many Unix/Linux editors such as `gedit` and `emacs` create a "backup file" when saving a file that already exists. The backup file's name is the same as the original file's name, except with a `~` character at the end. For example, when re-saving a file `midterm.txt`, the editor will create a file `midterm.txt~` with the previous contents of the file.

Write a complete Bash script that examines the files in the current directory, and if it finds any file(s) with a larger corresponding backup file, replaces the original file with the backup version. For example, if `midterm.txt` is 10000 characters in length and `midterm.txt~` is 12345, then `midterm.txt~` will replace `midterm.txt`. But if `midterm.txt` had been 20000 characters in length, it would be unchanged by your script. The replacement is a copy operation, not a move; the backup files themselves are not modified by your script.

If the current directory contains a given backup for which the original file no longer exists, you should not make any replacement. For example, if there is a file `gone.doc~` but no `gone.doc`, no change should occur.

Your script does not need to worry about errors; that is, you may assume that every file read/write/move/copy operation you perform will succeed. You may also assume that no file's name ends with more than one `~` character. Your script can create temporary files if you like, but they should be removed by the time your script is done. (It is possible to solve the problem with no temporary files.)

Hints: You counted lengths of files in one of the homework assignments. Also, in scripts it is easier to add something to a string than to try to remove something from it.

---

4. C Pointers

Suppose that the following C variables have been declared and assigned:

```
int x = 10;
int* y = &x;
int* z = (int*) malloc(2 * sizeof(int));
int a[2] = {20, 30};
z[0] = 40;
z[1] = 50;
```

Write the values of the following expressions in the table below. Write integers in decimal notation and pointers/addresses in hexadecimal notation. A few values are filled in for you; based on these, you should figure out the rest. Assume that the code is running on a 32-bit system, and that the stack grows downward in memory as shown in the examples in our slides. If an expression is likely to represent an illegal memory location, write `SEGFAULT`.

There is an exact correct answer for each box, but *we will be somewhat lenient about memory addresses* as long as you are pretty close to where the address should be.

x		y		z	**0x003e0000**
&x	**0x802f0038**	&y		&z	
*x		*y		*z	

a[0]		a[1]	
&a[0]		&a[1]	
*a[0]		*a[1]	

Now suppose that the following statements are executed:

```
*y = a[0];
a[1] = *z;
z++;
y = a;
```

Write down the new values of the same expressions. If a value is unchanged, you may leave it blank.

x		y		z	
&x		&y		&z	
*x		*y		*z	

a[0]		a[1]	
&a[0]		&a[1]	
*a[0]		*a[1]	

5. C Programming

Write a C function named `stutter` that accepts three parameters: An array of strings, an integer representing the size of the array (the number of strings in the array), and an integer representing a "stuttering factor." Your function should replace each string in the array with itself repeated the given number of times. If the stuttering factor value passed is 1 or less, the array should be unmodified. For example, suppose you have an array of strings `a` that contains the following elements:

- `a[0]` stores `"hello"`
- `a[1]` stores `"booyah"`
- `a[2]` stores `"Ford Prefect"`
- `a[3]` stores `"9 0 2 1 0 "`

The call of **`stutter(a, 4, 3);`** would stutter each string 3 times, yielding the following result after the call:

- `a[0]` stores `"hellohellohello"`
- `a[1]` stores `"booyahbooyahbooyah"`
- `a[2]` stores `"Ford PrefectFord PrefectFord Prefect"`
- `a[3]` stores `"9 0 2 1 0 9 0 2 1 0 9 0 2 1 0 "`

Assume that the strings in the array are stored in heap memory blocks sized exactly to fit their contents. All of the strings passed in the original array are null-terminated, and all of the resulting strings in the array after your function should be null-terminated as well. You may assume that the array passed to your function is not null, that none of the strings in the array are null, and that no pointers passed to you point to invalid memory locations. You may assume that the size (second parameter) is at least 0. You may assume that there is enough available memory on the system to accommodate any memory allocation you may need to do. Your function should not leak any memory; if you remove any regions of memory from use, you must free them.

Note: You can earn up to 1/2 credit if you solve this problem only for a stuttering factor of 2.
