CS 2413 Final

1. [16] Suppose a Perl program, `fern`, is invoked with the command:

```
fern -czv -f file1 file2 file3
```

and suppose `file1` contains the lines "`dir1`" while `file2` contains "`dir2`", "`dir3`" and "`dir4`" and `file3` contains "`dir5`" and "`dir6`".

At the start of the program:

(a) What is the list value of `@ARGV`?

(b) What is the value of `$#ARGV`?

(c) What value is assigned to `$val`?

```
$val = @ARGV;
```

(d) After the following statements have been executed,

```
$a = shift;
$b = pop;
$c = shift;
$d = <>;
```

i. What is the value of `$a`?

ii. What is the value of `$b`?

iii. What is the value of `$ARGV`?

iv. What is the value of `@ARGV`?

(e) If, following the above instructions, the following instructions are executed:

```
$b = <>;
$c = <>;
@d = <>;
```

i. What is the value of `@d`?

2. [20] Write a Perl script, called `split`, which will break a given file into files with a fixed number of lines. Both the name of the file and the number of lines per file (with the exception of the last file) will be given on the command line. The output files will have the same name as the original file but with a numbered extension added to the end (.0, .1, ..., .23, ...).

Thus an invocation which will split the file `file.txt` into chunks of 100 lines creating the files `file.txt.0`, `file.txt.1`, ... would be
`split file.txt 100`

3. [20] A system administrator wants to automatically search the text files under a number of directories for various regular expressions. Thus he has set up several files that contain the names of directories under which he wishes to search. Write a Perl script, `pgrep`, which will read these files, whose names are given on the command line, and will search all **TEXT** files which are under any of the directories in those files for the regular expression, given as the first argument on the command line. A sample invocation (`system` and `users` are files containing directories):

```
pgrep 'r00t' /etc/security/system /etc/security/users
```

4. [20] Intruders into a computer system typically install a root kit which replaces a number of binaries with trojaned binaries and cleans up all of the various log files. You want to know when this happens and which files were changed so you have created a file, `files.md5`, which contains a list of the all files followed by some white space and an MD5 signature of that file. Periodically you will run a program that creates a file, `files.new.md5`, which contains a list of the all current files and their MD5 signatures. Write a perl script which will compare both files and report the names of files that are:

   - In `files.md5` and not in `files.new.md5`
   - In `files.new.md5` and not in `files.md5`
   - Files whose MD5 signature has changed.

   Sample output (assume directory names have no white space):

```
/etc/hosts.deny deleted
/var/tmp/.ssh/passwords added
/etc/shadow MD5 signature changed
```

   A sample entry in either of the files looks like:

```
/etc/shadow $kDi.$ksKNfle14kNq$dp.KDMZw8/$u0e3
```

5. [25] Write a C **function** with prototype:

```
int delrecords(unsigned int rnum, unsigned int num, char *file, unsigned int rsz);
```

   A file can be considered to be a sequence of record of a fixed size, say `rsz` where the first `rsz` bytes are to be considered record 0, the next `rsz` bytes are record 1, ... The function `delrecords` will delete the `num` records starting with record number `rnum` in the file, `file`, moving all undeleted records after `rnum` up `num` records. The function must both open and close the file. If successful, `delrecords` will return a 0 else `delrecords` will return -1. The function should use low-level I/O.

5. [25] Write a C **function** with prototype:

```
int insblk(char *file1, unsigned int num1, unsigned int num2, char *file2,
           unsigned int num, char *fileout, unsigned int rsz);
```

A file can be considered to be a sequence of record of a fixed size, say **rsz**, where the first **rsz** bytes are to be considered record 0, the next **rsz** bytes are record 1, ... The function **insblk** will read records num1 to num2 inclusively from file1 and insert them before record num in file2 writing all records into a new file called fileout. The records must be read/written one at a time but you may assume that they are less than 4096 in size. The function must also both open and close the files. If successful, **insblk** will return a 0 else **insblk** will return -1. The function should use low-level I/O.

6. [25] Implement a low-level popen, called ppopen. The prototype is:

```
int ppopen(const char *command, const char *type);
```

The function ppopen will fork and exec command after having created one pipe. If type is "r" then the pipe will be set for command to write back to the parent process via stdout while if type is "w" then the pipe will be set for the parent process to write over the pipe to command's stdin. The function ppopen will return the appropriate file descriptor of the pipe so that the parent can communicate, in the appropriate direction, with command.

7. [25] Write a **function**, called compose, that will be passed two structures of the form:

```
struct command
  {
    char *cmd;       /* name of command */
    char *argv[10]; /* Null terminated argv list of arguments to command
                       (including command)*/
  };
```

The function compose() with prototype:
`int compose(struct command cmd1, struct command cmd2);`
will fork and exec both commands, piping the output of the first command into the second command. The function **returns** 0 on success and -1 on error.

8. **[25]** Write a C program which will create a total of 33 processes all of which will have their own separate pipe i.e. all processes will be able to write to all other processes but each process will only read from its own pipe. Each process will be assigned a unique index 0, 1, ..., 32. Each process will write it's index and pid (you define the message structure) to every other process (but not to itself). Then each process will read all the messages written to its pipe and print on stdout, for each message, a line similar to:
Process 35894 index 24 heard from process 35992 index 17
Be careful that your code doesn't hang!

9. **[30]** In the system that you are writing, many programs (called clients) will be executing and will need other programs, called servers, to process some data. In this version of the system each program will write to a well known fifo (named pipe) with the name "`/tmp/dispatch`". Each dispatcher program repeatedly reads a message from the fifo, execs the requested server (program) after connecting the pipes. The server is expecting its data via stdin and will write its data to stdout. The dispatcher must be certain that the server's stdout is redirected to the return fifo named in the message and that the server's stdin is redirected to the data fifo named in the message. It is the clients responsibility to write the data to the data fifo. The format of the message is:

Data	Field (first byte - last byte)
program name (null terminated)	0-127
return fifo name (null terminated)	128-255
data fifo name (null terminated)	256-383

Write the dispatcher program (in C) for this system assuming that there may be several dispatcher programs executing at the same time. There should be no long-lived zombies created nor should the dispatcher wait for a potentially long time. The dispatcher should follow the current path (bad idea) to find the program when exec'ing.

Note that a dispatcher does not terminate but constantly reads messages from the dispatch fifo.