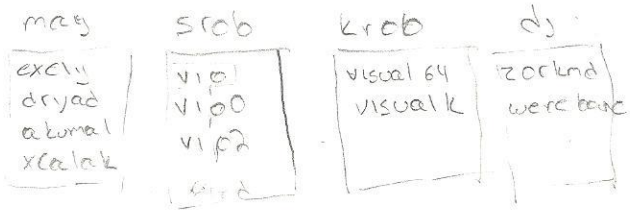
CS 3423 Final

1. [20] Suppose a Perl program, spam, is invoked with the command:

```
nettrace -l maynard srobbins krobbins dj
```

and suppose maynard contains the lines "exely", "dryad", "akumal" and "xcalak", srobbins contains "vip", "vip0" and "vip2", krobbins contains "visual64" and "visualk" while dj contains "zorkmid" and "werebane".



At the start of the program:

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

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

(c) In the following what value is assigned to $size:

```
$size = @ARGV - 1;
```

(d) Consider the execution of following initial statements assuming the original command,

```
$a = shift;
$b = pop;
$b = shift
push(@ARGV,$b);
$d = <>;
```

i. What is the value of $b?

ii. What is the value of $d?

iii. What is the value of $ARGV?

iv. What is the list value of @ARGV?

(e) On the other hand if the following statements were to be executed first:

```
$a = shift;
$b = shift;
@d = <>;
$e =  pop @d;
@f =  splice(@d,2,3,"fred");;
```

i. What is the value of $e?

ii. What is the value of @d?

iii. What is the list value of @f?

2. [20] Write a Perl script, called heads, which will, for each file on the command line, print to the output file the name of the file followed by a colon and a newline followed by the first 5 lines of that file. The output file name is the last token on the command line.

```
heads file1 file2 file3 file4 outfile
```

3. [20] Write a Perl script, `pgrep`, which will search all **TEXT** files which are under any of the directories whose names are in the files whose names are on the command line. The search is to search the contents of these files for the regular expression which is given as the first argument on the command line. For each line in which a match is found, the pathname of the file is to be printed followed by a colon and then the matching line. A **sample** invocation:

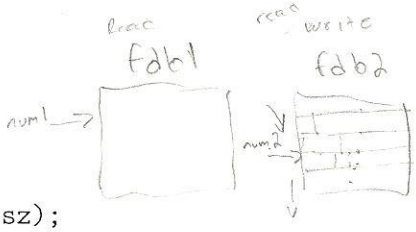
```
pgrep 'ca*t+' file1 file2 file3 file4
```

4. [25] Student project reports are stored in files, named with the student's usernames, which are in project directories under the current directory. For example, the project report for maynard's project3 is in the file `project3/maynard`. Each project report will have a single line, among the many lines in the report, which starts with "Total Points:", then some white space followed by that students score for that project. This line looks like:

```
Total Points:      27
```

Write a perl script that is to be run in the current directory which contains the project directories. The script should compute the total project grade for each of the students and then should print out the usernames, each followed by a tab and the total project grade for that user. You can, for a 10 point reduction, do this only for the user `maynard`. A `sample` invocation: The arguments are the project directory names:

```
pgrades project2 project3 project4 project5 project6 project7
```



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

```
int reccopy(int num1, char *fdb1, int num2, char *fdb2, int rsz);
```

The function `reccopy()` will copy the record number `num1` from the file `fdb1`, **inserting** the record into the file `fdb2` at record position `num2`. Pay careful note to the fact that you are to insert the record and **not** overwrite a record in `fdb2`. `fdb1` should remain unchanged.

A file can be considered to be a sequence of records 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 records **must** be read/written one at a time and you may assume in this problem that they are less than 4096 in size. The function must also both open and close all files. If successful, `insert` will return a 0 else `insert` will return -1. The function must use low-level I/O.

6. [25] You are running `portsentry` on your machine which detects probes at various ports and will block the prober. You want a program which will report the attacks which occured after the last report. The last report was based on the file which is currently named `/var/log/ps.udp.1` while the current set of attacks (which includes the old attacks) is stored in `/var/log/ps.udp.0`. Write a C program that will do a diff between these two files and email root the results. This could be written on the command line as below (but don't).

```
diff /var/log/ps.udp.1 /var/log/ps.udp.0 | mail -s Attack root
```

7. [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 == "r" then the pipe will be set for command to write back to the parent process via stdout while if type == "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 so that the parent can communicate, in the appropriate direction, with command.

8. [25] Write a C program which will create a **total** of 35 processes all of which will have their own separate pipe, i.e. all processes must 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, ..., 34. Each process **with an odd pid** will write it's index and pid to every other process (but not to itself). The processes with even pids will not write any messages. Then each process (odd and even) 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. [25] 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". This fifo already exists in the file system. The dispatcher program reads each message from the fifo, execs the appropriate server 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 dispatchers wait for a potentially long time. The dispatchers should follow the current path (bad idea) to find the program when exec'ing.