UCLA Computer Science 35L Final Exam - Winter 2019
Open book, open notes, closed computer.
100 points total, 180 minutes total, 1 point = 1.8 minutes
Each top-level question is 10 points and should take 18 minutes.
Write answers on the exam.

Name:_____

Student ID:_____

1	2	3	4	5	6	7	8	9	10	total

1. You are Dr. Eggert with username ‘eggert’ and group ‘csfac’, and are trying to review some basic concepts. Your current working directory is '~/exam/'.

There are two files you are particularly interested in: foo and bar. The following are the contents and permissions on those files. (The flag -i prints the file's file inode number.)

```
$ cat part1/foo
This is the finals for CS35L
I am foo
$ cat part2/bar
Checking hard and symbolic links
I am bar
$ ls -li part1/
total 0
3165230 -rw-r--r-- 1 eggert csfac 38 Mar 15 02:37 foo
$ ls -li part2/
total 0
3165232 -r--r--r-- 1 eggert csfac 42 Mar 14 14:35 bar
```

You execute the following commands :

```
$ ln part1/foo baz
$ ln -s part2/bar qux
```

Answer the following questions based on the information above. If you think there is any error while executing the commands above or the commands mentioned later, you can briefly describe the error.

1a (1 point). What do the above commands mean?

1b (1 point). Write a shell command or commands to count the total symbolic links from the current directory up to 2 levels of directories below the current directory. (Hint : 'find -maxdepth N' descends at most N directory levels below the command line arguments.)

1c (2 points). What will be the inode (index-node) of the files baz and qux respectively?

1d (2 points). After executing the below commands, what will be the contents of the files baz and qux?

```
$ mv part1/foo part1/foobar
$ mv part2/bar part2/barfoo
```

1e (2 points). You execute the commands in step (d). What will be the content of the files baz and qux after executing the below commands?

```
$ rm part1/foobar
$ rm part2/barfoo
```

1f (2 points). For this question assume that we do not execute the commands in parts (d) and (e).

What will be the output of the below mentioned commands?

```
$ echo “Updating files” >> baz
$ echo “Updating files” >> qux
$ cat part1/foo
$ cat part2/bar
```

2. We have a directory with many genetic data files, each named like this:

    FirstnameLastname.gene_ID(chromosome_number)

FirstnameLastname contains only alphabetic characters (no spaces). Firstname and Lastname both begin with one capitalized letter, and both Firstname and Lastname must be at least 2 letters long. The gene_ID is a combination of letters and digits and chromosome_number is a number from "1" to "22". E.g.:

  JoeBruin.ENSG00000112137(6)

A FirstnameLastname pair may appear in multiple files, and a gene_ID may appear in multiple files.

2a (3 points). Assume that we decide to list the contents of the directory and store the result in a variable. Which of the following would successfully accomplish this (if any)? Explain your reasoning for each. (Assume that the directory path is stored in the variable GDIR).

  A. LIST=`ls $GDIR`
  B. LIST='`ls $GDIR`'
  C. LIST="`ls $GDIR`"

2b (2 points). Assume for this part, that one of the above commands worked, and that we now have the results of the ls command in LIST. Write a Bash script (you may skip the shebang line) that outputs to stdout every filename in LIST that does not have read permission granted. (Note: the "-r FILE" option of the test command returns true if FILE exists and read permission is granted.)

2c (1 point). Now assume that we did not do the steps in A and B. Instead, we have decided to run our ls command and pipe it into a grep command. Given our specific problem, would it be more advantageous to use Basic Regular Expression, or Extended Regular Expression? Briefly explain.

2d (4 points). Using whichever mode (BRE or ERE) you chose in part C, write a regular expression that could be used in a grep command to output only the files that meet the following criteria. Lastname begins with an "R", "C", "S", or "Q". The gene ID begins with "ENSG" followed only by 1 or more numerical digits. Lastly, the chromosome number has at least 2 digits.

3a. A Makefile has the following contents:

```
move : car
car : fuel.o wheels.o car.o
        g++ -o car fuel.o wheels.o car.o
wheels.o : wheels.cpp wheels.h
        g++ -c wheels.cpp
fuel.o : fuel.cpp fuel.h
        g++ -c fuel.cpp
car.o : car.cpp wheels.h fuel.h
        g++ -c car.cpp
clean :
        rm -f wheels.o fuel.o car.o car
```

Assume that all the header files are present in the same directory as the Makefile. Look at the contents of the Makefile thoroughly and answer the following questions:

3a1 (2 points). What happens when you run the command ‘make all’ on the terminal? Explain.

3a2 (1 point). Why do we need makefiles? List any two distinct reasons.

3b (2 points). The first two lines of a patchfile (exam_patch.patch) are as follows:

```
---abc/cs35l/Documents/exam.c
+++xyz/cs35l/Documents/exam.c
```

Given that exam_patch.patch is present in the cs35l directory, write the appropriate patch command to update exam.c if the current working directory is Documents.

3c (5 points). A Python 3 file (armstrong.py) has the following statements:

```
    def numDigits(n):
        #Write your code here

    def isArmstrong(n):
        #Write your code here

    n = int(input())
    print("Is", n, "an Armstrong number?", isArmstrong(n))
```

Complete this Python 3 file by defining the isArmstrong(n) and numDigits(n) functions such that the Python script does not throw any errors when it is run. Throw an appropriate exception for negative numbers. It is guaranteed that the input is an integer.

Note:

a) isArmstrong(n) should return "Yes" or "No".

b) An Armstrong number of x digits is a positive integer such that the sum of the nth power of its digits is equal to the number itself.

Examples:

153 is an armstrong number because 13 + 53 + 33 = 153 (the number itself).

Similarly, 1634 is an Armstrong number too since 14 + 64 + 34 + 44 = 1634.

4. Look at the code snippet below and write what gets printed at every line. Use underscores for spaces:

```
  #include <stdio.h>
  char *c[] = {"the", "quick brown fox", "jumped", "over the", "lazy dog"};
  char **cp[] = {c+3, c+2, c+1, c, c+4};
  char ***cpp = cp;

  int main(void)
  {
      printf("%s\n", **(cpp+3));   //Line 1
      printf("%s\n", **(cp+4)+4);  //Line 2
      printf("%s\n", **(++cpp));   //Line 3
      printf("%s\n", **cp);        //Line 4
      printf("%s\n", **(++cpp)+5); //Line 5
      return 0;
  }
```

4a (2 points). Line 1 output:

_____

4b (2 points). Line 2 output:

_____

4c (2 points). Line 3 output:

_____

4d (2 points). Line 4 output:

_____

4e (2 points). Line 5 output:

_____

5. Consider the following program and two text files:

question.c:

```
#include <fcntl.h>
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
int
main(void)
{
  int fd1,fd2,sz1,sz2;
  char *c1 = (char *) calloc(100, sizeof(char));
  char *c2 = (char *) calloc(100, sizeof(char));

  fd1 = open("content1.txt", O_RDWR);
  fd2 = open("content2.txt", O_RDONLY);

  if (fd1 < 0){
     perror("Error in opening content1.txt");
     exit(1);
  }
  if (fd2 < 0){
     perror("Error in opening content2.txt");
     exit(1);
  }
  sz1 = read(fd1, c1, 12);
  sz2 = read(fd2, c2, 8);
  printf("fd1 = %d, fd2 = %d \n", fd1, fd2);
  printf("Called read which returned %d \n", sz1);
  printf("Content read: %s \n", c2);
  write(fd2, c1, sz1);
  write(fd1, c2, sz2);
  close(fd1);
  close(fd2);
}
```

content1.txt:

```
Lord_of_the_Rings
```

content2.txt:

```
Charlie_Chaplin
```

Each text file consists of just one line; underscores (_) denote spaces, and the text lines do not have leading spaces.

5a (1 point). Will the above program execute? If yes, what is the output of the program? If no, why would it not execute?

5b (1 point). What are the contents of content1.txt and content2.txt post execution of the above code? (Rewrite the original contents of the file if you said that the program would not execute.)

Let’s say I change the open functions in this problem as given below. (The rest of the code remains unchanged.)

```
fd1 = open("content1.txt", O_RDWR | O_APPEND);
fd2 = open("content2.txt", O_RDWR);
```

After execution of the modified program on the original data:

5c (1 point). What are the contents of content1.txt?

5d (1 point). What are the contents of content2.txt?

5e (2 points). What functions would you change in the above programs to move the program from using system calls to inbuilt library functions? Mention both the functions you would change and their equivalent replacements.

5f (2 points). Comment on the performance of the code if you had written it using the library functions you listed in part (e). Which would be faster and why?

5g (2 points). Rewrite the above code to print the entire contents of content1.txt and content2.txt using system salls. Do not use functions printf(), puts(), etc.

6. Parallel Computing & Synchronization

6a (2 points). Overhead of mutex lock and unlock. In the table below, you are given 3 different snippets of using a mutex to solve critical sections. Assume they are running on the same machine. Please give the rank of their running time (e.g. “A takes more time than B, which takes more time than C”) and briefly explain why.

```
/* A */
int N = 10000000;
pthread_mutex_lock(&m);
for (int i = 0; i < N; i++)
  sum++;
pthread_mutex_unlock(&m);

/* B */
int N = 10000000;
for (int i = 0; i < N; i++) {
  pthread_mutex_lock(&m);
  sum++;
  pthread_mutex_unlock(&m);
}

/* C */
int N = 10000000;
int local = 0;
for (int i = 0; i < N; i++)
   local++;
pthread_mutex_lock(&m);
sum += local;
pthread_mutex_unlock(&m);
```

6b. Multithreaded time. Here are the outputs of the 'time' command when the multi-threaded version of the sort program above was executed on the 16 core server.

```
Threads used : 1 real 0m31.698s user 0m31.607s sys 0m0.018s
Threads used : 2 real 0m16.403s user 0m31.796s sys 0m0.017s
Threads used : 4 real 0m10.803s user 0m31.750s sys 0m0.017s
Threads used : 8 real 0m7.012s   user 0m32.989s  sys 0m0.015s
```

Please answer the following questions:

6b1 (1 point). What could be a possible reason for the 'user' time being almost the same in all the cases?

6b2 (1 point). Why is the 'user' time greater than the 'real' time in some of the cases?

6b3 (1 point). What could happen to the above times, if the server administrators set a policy that allowed your programs to use at most 2 of the cores?

6c (5 points). Multithreaded Sort. You are required to sort a set of 1 billion integer scores in descending order. This can take a very long time if your program is single threaded. Your task is to write a multi-threaded program, using pthreads, that distributes the workload across a set of 16 threads. Some starter code is given below. The standard library qsort() function, with a compare() function, should be used to do the sorting per thread. You don't have to rewrite anything that is already given below.

```
#include <stdio.h>
#include <math.h>
enum { SCORE_COUNT = 1000000000 };
enum { NTHREADS = 10 };

//Your code here [1]

int main(int argc, char** argv) {
  int *scores = (int*) malloc(sizeof(int) * SCORE _COUNT);
  if(!scores) {
    fprintf(stderr, “Cannot allocate memory!\n”);
    exit(-1);
  }

  // Assume the following function call loads values into 'scores'
  LoadScores(scores);


  // Assume the following function call prints the sorted values
  PrintScores(scores);
  free(scores);
  return 0;
}
```

7. You have maintained a C project. The below is the output of ls command when applied on your project directory:

```
$ ls
main.c    foo.c    bar.c    foo.h    bar.h    type.h
```

And you know that

foo.h includes type.h

main.c includes foo.h and bar.h

bar.c does not include any header files.

7a (6 points). Write a Makefile that:

A. When you type make, it will generate an executable called criu compiled from main.c, foo.c and bar.c.

B. When you change one of the .c or .h files, only minimal number of the .c files got recompiled and the newly created hello can correctly reflect your change.

C. When you type make clean, it will remove all the intermediate file generated by the compilation process.

D. When you type make tarball, it will generate a bzip2 compressed file named clean.tar.gz that contains all the .c file .h file listed by the ls command and the Makefile itself.

7b (4 points). A buggy Makefile may overwrite your own source code! For example, consider the following Makefile that compiles hello.c to executable hello:

```
default: hello.c
        gcc -o hello.c hello.c
```

When you type 'make', the hello.c will be overwritten with the hello.c executable.

You are afraid that there may be potential bugs in the Makefile written in part (a) which will overwrite the existing source code.

Add a new target called protection into the Makefile you have written in part (a) to prevent Makefile from overwriting your source code files. Once you have typed make protection in the terminal to execute the target, even if there is a bug in the Makefile that will overwrite the source code files, the corresponding target will fail.

You are not allowed to move any files outside the current directory or create any new files.

Write down the protection target and related target (if any) below.

8. This question has multiple-choice subproblems.
For each such problem, write the single best answer,
which will be (i), (ii), (iii), (iv), or (v).

8a (2 points).

(i). If you don't trust your server, you cannot use OpenSSH to connect to it, as it can easily corrupt your client.
(ii). If you don't trust your client, you can still use OpenSSH to connect to a trusted server.
(iii). If you don't know the name or IP address of your server, you can use OpenSSH to discover this info in a secure way.
(iv). If you don't know the name or IP address of your client, you can still use OpenSSH to connect to a server.
(v). If you don't trust your network, you can still use OpenSSH to discover whether your server is running.

8b (2 points).

(i). ssh-agent improves security by making a copy of private keys.
(ii). ssh-agent acts on your behalf by running on the server and executing commands there, under your direction.
(iii). Even if the attacker surreptitiously replaces the ssh-agent program with a modified version, your communications will still be secure.
(iv). ssh-agent eliminates all need for password authentication when communicating to the SEASnet GNU/Linux hosts.
(v). If you successfully use a typically-configured ssh-agent and then log out from the client and then log back in again, you can then connect to the same SSH server again without typing any additional passwords or passphrases.

8c (2 points).

(i). OpenSSH typically uses public-key encryption for authentication, because private-key encryption is less secure.
(ii). OpenSSH typically uses private-key encryption for data communication, because public-key encryption is less efficient.
(iii). When you run 'ssh', it chooses its authentication key randomly from a large key space, to make eavesdropping harder.
(iv). The OpenSSH client and server are essentially symmetric, so that it's easy and common to use the same program as either a client or a server.
(v). Once your private keys are 1024 bits long, there's no point making them any longer, as they're impossible to break.

8d (2 points).

(i). OpenSSH is not limited to just one client-server connection; for example, a team of five people can use OpenSSH to communicate information to each other.

(ii). For security, OpenSSH refuses to connect to programs written by other people; for example, a client running the OpenSSH code will connect only to a server running the OpenSSH code.

(iii). Although port forwarding can be used to display from an OpenSSH server to an OpenSSH client, the reverse is not possible: you cannot use port forwarding to display from an OpenSSH client to an OpenSSH server.

(iv). For security, port forwarding cannot be chained: that is, you cannot ssh from A to B, and then from B to C, and use port forwarding to let a program run on C and display on A.

(v). When using port forwarding when connecting to SEASnet, one should take care not to create a forwarding loop, as this can lead to a cycle of packets endlessly circulating on the Internet.

8e (2 points).

(i). It’s not a good idea to connect to a SEASnet GNU/Linux server and use GPG on the server to sign a file, because then an attacker on the network can easily snoop your GPG passphrase.

(ii). A detached signature file must be protected as securely as the private key it’s based upon; otherwise an attacker will be able to forge your signature more easily.

(iii). When generating a key pair it’s important to use a private entropy pool on SEASnet, not the shared pool that everybody can access, because otherwise an attacker on SEASnet might be able to guess your key more easily by inspecting the public entropy pool.

(iv). Exporting a GPG public key to ASCII format neither improves nor reduces its security.

(v). Because a detached cleartext signature isn’t encrypted, it is easily forged by an attacker with access to the file being signed.

9. Ava and Max are working on a project, using git as a versioning tool. The project is in a repository called Asymmetry, and is stored locally, since it is just a small project. Ava has checked out from the master into a branch called issue42, and Max into a branch called issue49 (both these branches are set to track the master) The state of the repo at this time is that it has three commits, arranged as follows:

  c41fa is the oldest (original) commit.
  8ac11's parent commit is c41fa.
  6ab77's parent commit is 8ac11.
  6ab77 is tagged by 'master', 'issue42', and 'issue49'.

At the commit 6ab77, the repository has the following files:

  Application.py
  dbconn.py
  frontend.py
  index.html
  README.md
  structure.css

Ava, in her branch issue42, creates two files - format.py and clean.py; modifies frontend.py, ensures her code works as expected, and then runs the following commands:

  git status
  git add format.py frontend.py
  git clean --force
  git commit -m "issue42: adding prettify patch"
  git checkout master
  git merge issue42 (this merges the code with the master branch)
  git tag -a iss42 -m "commit for issue42"

While Ava is working on her fixes, Max is also developing his end of the code, on his branch. He modifies dbconn.py and creates test_data.csv and testpage.html, tests his code, and runs the following commands:

  git status
  git add dbconn.py test_data.csv
  git commit -m "issue49: testcases for the database"

However, he does not run the merge commands:

  git checkout master
  git merge issue49

until the next day, by which time Ava is done with her development (and git commands).

[continued on next page]

9a (1 point). What command(s) should Max run before running the checkout & merge commands above to ensure that he has (and tests) the latest version of the code from the master, including Ava's? Assume he knows there will be no conflict with their files. Select any one:

A. git fetch
B. git fetch && git status
C. git fetch && git merge
D. git revert

9b (3 points). List the files in the repository (in the master branch) once Ava, Max have pushed their code. Indicate which file(s) are modified and which have been added.

9c (3 points). A week or so later, QA finds a problem with the way Ava has fixed issue 42. She has to fix it again, but unfortunately, her laptop disk has been reformatted since then and she's lost all her local changes. What's the best way Ava can get her specific fix back and work on it? Specify the command(s) for the same.

9d (3 points). There is also an issue, it is later discovered, with the way Max has written some of his testcases. Unfortunately, he hadn't Ava's foresight when he did the git push, so he must do things differently. How can he go about retrieving his commit ID, given that he remembers that he put the issue ID (issue49) first, and the word "testcases" somewhere later, in his commit message? (Use any git and/or Linux commands necessary.)

10 (10 points). Consider the following ACM TechNews summary of an article published in the New York Times on March 14. Explain the relationship between this article's topic and the ACM TechNews topic that you summarized in your solution to Assignment 10. Or, if the two topics are completely unrelated, explain why they are unrelated.

"Facebook's Daylong Malfunction Is Reminder of Internet's Fragility"

Facebook said it has corrected a technical error that caused a nearly 24-hour-long service interruption for Instagram, WhatsApp, Messenger, and other Facebook properties this week. According to a Facebook spokesperson, a "server configuration change" had a cascading effect throughout the company's network, triggering a recurrent loop of problems that kept escalating. The incident serves as a reminder that the Internet can still be hobbled by human error. For years, Facebook has recruited engineers on the idea that, within weeks, they can release computer code that reaches billions of people, especially as the company devises a strategy to consolidate the infrastructure of its "family of apps." However, the outage demonstrated that the more tightly intertwined a network becomes, the more likely a small technical problem caused by a single employee can have far-reaching consequences.

CS 35L Winter 2019 Final Exam Solutions

Question 1.

1a) Creating hard link and soft (symbolic) links

1b) find . -maxdepth 2 -type l | wc -l

1c) baz = 3165230 , qux = cannot be determined from the data , will not be 3165232

1d) mv does not affect hard link. Hence baz will still have the same contents as before. Same as that of foo.

mv changes the soft link , hence qux will have a broken link and error while opening it (no contents)

1e) rm does not affect hard link. Hence baz will still have the same contents as before. Same as that of foo.

rm changes the soft link , hence qux will have a broken link and error while opening it (no contents)

1f) foo will be appended with the new contents of baz.

bar does not have write permissions as can be seen from the question screenshot. Hence nothing can be written to qux as well. echo "Updating files" >> qux will give error.

Question 2

2a. A. This will work because backticks are used.

B. This will not work, because single quotes do not preserve the meanings of the back ticks.

C. This will work, because double quotes preserve the meanings of the back ticks.

2b.

```
for FILE in "$LIST"
 do if [ ! -r "$FILE" ]
then
 echo $FILE
fi done
```

2c.

Using a basic regular expression will work because we search for the "-r" suffix which indicates a read permission.

An extended regular expression will also work.

2d.

You can assume that apart from the first letter of the firstname and lastname, the rest of the firstname and lastname is in lower case.

```
`^[A-Z][a-z]+[RCSQ][a-z]+ \. ENSG[0-9]+\([0-9]{2,}\)`
```

Question 3:

3a1) The Makefile does not contain a target named “all”. Hence when make all is typed on the terminal, it results in an error. make move on the other hand would have worked fine.

3a2)

i) In large projects, time-consuming re-compiles are avoided by makefiles.

ii) Maintaining dependencies across different files in a project becomes less cumbersome with the help of makefiles.

3b) patch -p3 < ../exam_patch.patch

3c)

```
def numDigits(n):
      return len(str(n))

def isArmstrong(n):
      if n < 0:
            raise ValueError("Please input a
non-negative integer!")
      numDig = numDigits(n)
      tot = 0
      for i in str(n):
            tot += int(i) ** numDig
      if tot == n:
            return "Yes"
      else:
            return "No"

n = int(input())
print("Is", n, "an armstrong number?",
isArmstrong(n))
```

Question 4:

4a. the

4b. _dog

4c. jumped

4d. over_the

4e. _brown_fox

Question 5:

5a)

Yes, the program will execute. The output is:

fd1=3, fd2 = 4

Called read which returned 12

Content read: Charlie_

5b) content1.txt = Lord_of_the_Charlie_

content2.txt = Charlie_Chaplin

5c) content1.txt = Lord_of_the_Rings

Charlie_

5d) content2.txt = Charlie_Lord_of_the_

5e) Open -> fopen()

Close -> fclose()

Read -> fread()

Write -> fwrite()

5f) Expecting an answer on the lines of buffered and unbuffered I/O

5g)

```
int main(){
int fd1,fd2,sz1,sz2;
char *c1 = (char *) calloc(100, sizeof(char));
char *c2 = (char *) calloc(100, sizeof(char));
fd1 = open(“content1.txt”, O_RDWR);
fd2 = open(“content2.txt”, O_RDWR);
if (fd1 < 0){
perror(“Error in opening content1.txt”);
exit(1);
}
```

```
if (fd2 < 0){
perror("Error in opening content2.txt");
exit(1);
}

sz1 = read(fd1, c1, 20);
sz2 = read(fd2, c2, 20);
write(1, c1, strlen(c1));
write(1, c2, strlen(c2));
close(fd1);
close(fd2);
}
```

Question 6.
Omitted as it's on threading.

Question 7.
7a.

```
all:
criu
criu: main.o foo.o bar.o
 gcc -o criu main.o foo.o bar.o

main.o: main.c foo.h bar.h type.h
 gcc -o main.o main.c

foo.o: foo.c
 gcc -o foo.o foo.c

bar.o: bar.c
 gcc -o bar.o bar.c

clean:
 rm ./*.o

tarball:
 tar -cjvf clean.tar.gz main.c foo.c bar.c
bar.h type.h foo.h MakeFile
```

7b.

```
protection:
chmod 444 ./*.c
chmod 444 ./*.h
make criu
chmod 744 ./*.c
chmod 744 ./*.h
```

Question 8.
8a) iv 8b) i 8c) ii 8d) i 8e) iv

Question 9
i. C
ii.

```
Application.py
dbconn.py*
frontend.py*
index.html
README.md
structure.css
format.py+
test_data.csv+

*: modified
+: added
```

iii.

```
git clone
git checkout iss42 -b "(any branch name)"
```

iv.

```
`git log | grep -B 10 "issue49.*testcases"`
```

(-B 10 gives the ten lines before the match, you would've got extra credit for using this flag, but no points lost for not using it)
OR `git log > git-log.txt`

open git-log.txt in

e-macs, use extended regexp search (C-M-s) to look for "issue49.*testcases" and locate the commit

Fall 2017 Zhaowei Tan Question #2

```
#!/bin/bash

fake_path='path1:path2:path3'

poly () {
  echo "$1" ' ' "$2"
  result=$(($1*$1*$1+$2*$2+$1*$2))
  echo 'The result is' "$result"
}

if [ ! "$1" ] || [ "$2" ]; then
  echo 'Only 1 argument allowed'
  exit 1
fi

if [ "$1" ]; then
  if [ ! -d "$1" ] || [ -L "$1" ]; then
    echo 'Argument is not a valid directory'
    exit 1
  fi
  directory="$1"
fi

echo 'directory is' "$directory"

path2=`echo $fake_path | tr ":" " "`
# OR: path2=${fake_path//:/ }
echo "$path2"
in_path=0
for a_path in $path2
do
  echo "$a_path"
  if [ $a_path = $directory ]; then
     echo 'match found'
     in_path=1
  fi
done
```

```
if [ "$in_path" == 0 ]; then
  echo 'match NOT found'
  fake_path+=:"$directory"
fi

file=`ls -prt | grep -v / | head -n 1`
x=${#file}
# y=0
# all_files=`ls`
# for a_file in $all_files
# do
#  if [ ${#a_file} -eq 2 ]; then
#    y=$((y+1))
#  fi
#done

#alternative method, one-liner
y=`ls | awk 'length == 2' | wc -l`

poly "$x" "$y"

echo 'fake path is' "$fake_path"
```

Final Exam: CS 35L Section 7

University of California, Los Angeles

Fall 2017

Name: _____

Student ID: _____

Question	Points
1	**/12**
2	**/16**
3	**/16**
4	**/16**
5	**/20**
6	**/20**
Total	

Instructions:

1. This examination contains 9 pages in total, including this page.
2. You have **three (3) hours** to complete the examination.
3. Write your answers in this exam paper. The draft papers will not be accepted.
4. You may use any printed resources, including lecture notes, books, slides, assignments. **You cannot use any electronics.**
5. Please finish the final **independently**. If you have any question, please raise your hand.
6. There will be partial credit for each question.
7. Best of luck!

Question 1: Concept Question

[12 pts] Choose **two (2)** of the following three questions and answer them concisely.

(a) What is cloud computing? What is the advantage (list at least two) of it?

(b) What are the different levels (distance to hardware) of programming languages? Why do we need each of them?

(c) Draw a simple diagram of how public-key encryption system works (what are the keys involved, how to use keys for encryption/decryption).

Average/Median/Max: 10.67/12/12
We do well in this question.

This is a relatively easy question. Copy the concepts from the slides and we receive the full credit.

Question 2: Unix Command

[16 pts]

The following is the subset of the man page for the "df" command.

```
DF(1)                    BSD General Commands Manual                    DF(1)

NAME
     df -- display free disk space

SYNOPSIS
     df [-b | -h | -H | -k | -m | -g | -P] [-ailn] [-t] [-T type]
        [file | filesystem ...]

LEGACY SYNOPSIS
     df [-b | -h | -H | -k | -m | -P] [-ailn] [-t type] [-T type] [file |
     filesystem ...]

DESCRIPTION
     The df utility displays statistics about the amount of free disk space
     on the specified filesystem or on the filesystem of which file is a
     part.  Values are displayed in 512-byte per block counts.  If neither
     a file or a filesystem operand is specified, statistics for all
     mounted filesystems are displayed (subject to the -t option below).

     The following options are available:

     -a      Show all mount points, including those that were mounted with
             the MNT_IGNORE flag.

     -b      Use (the default) 512-byte blocks.  This is only useful as a
             way to override an BLOCKSIZE specification from the environ-
             ment.

     -g      Use 1073741824-byte (1-Gbyte) blocks rather than the default.
             Note that this overrides the BLOCKSIZE specification from the
             environment.

     -i      Include statistics on the number of free inodes. This option
             is now the default to conform to Version 3 of the Single UNIX
             Specification (``SUSv3'') Use -P to suppress this output.

     -k      Use 1024-byte (1-Kbyte) blocks, rather than the default.  Note
             that this overrides the BLOCKSIZE specification from the envi-
             ronment.

     -l      Only display information about locally-mounted filesystems.

     -m      Use 1048576-byte (1-Mbyte) blocks rather than the default.

     -n      Print out the previously obtained statistics from the filesys-
             tems.  This option should be used if it is possible that one
             or more filesystems are in a state such that they will not be
             able to provide statistics without a long delay.  When this
```

```
        option is specified, df will not request new statistics from
        the filesystems, but will respond with the possibly stale sta-
        tistics that were previously obtained.

-P      Use (the default) 512-byte blocks.  This is only useful as a
        way to override an BLOCKSIZE specification from the environ-
        ment.

-T      Only print out statistics for filesystems of the specified
        types.  More than one type may be specified in a comma sepa-
        rated list.  The list of filesystem types can be prefixed with
        ``no'' to specify the filesystem types for which action should
        not be taken.  For example, the df command:

              df -T nonfs,mfs

        lists all filesystems except those of type NFS and MFS.  The
        lsvfs(1) command can be used to find out the types of filesys-
        tems that are available on the system.
```

Given the above instructions, you should specify the sequence of commands you use to finish the following:

(a) Use df to check the free disk space on the computer. The results should only include the locally-mounted filesystems, and exclude the autofs type filesystems. The display should use 1-Gbytes blocks.

(b) Use pipeline and find all the lines that include the keyword "**100%**" in the previous output. Save the extracted results as df.txt to your HOME directory.

(c) Go to your HOME directory. For df.txt, grant r/w/x permission to the owner, r/x permission to the group, and r permission to anyone.

(d) Change the filename to df_new.txt, create a new directory called test under HOME directory, and copy the file into this directory.

Average/Median/Max: 13.57/15/16
We do well in this question as well.

(a) Use the following three options:

- -l (for local filesystems)
- -T noautofs (for excluding autofs)
- -g (for 1-G blocks)

(b) We should use pipeline (|), 'grep', and redirection (>) in the answer. The Home directory should either be tilde or $HOME.

(c) The correct syntax is 'chmod number filename'. Some of us miss the filename.

(d) Note that the question asks us to copy the file, instead of moving it. The syntax for cp is 'cp filename location'.

Question 3: Bash Programming

[16 pts] Write a bash script "easy" to finish the following tasks:

(a) The script takes two command line arguments (i.e. one extra parameter except for your script name). The second parameter is the path of a directory.

(b) This script will check whether the input parameter is in the $PATH variable. If not, add this directory into the $PATH variable.

(c) Under this directory, find the oldest regular file and gets the length of its filename. Denote this length as x.

(d) Under this directory, find the number of files that is two-character long. Denote this number as y.

(e) Write a function which calculates $x^3 + y^2 + x \cdot y$. Call this function and print the result to the standard output.

(f) Your script should handle two exceptions and outputs "error": when the input parameter number does not satisfy the requirement, and when the second argument is not a valid directory.

Average/Median/Max: 10.53/11.75/15
The outcome of this question is not as good as I expected. All the answers could be found either in notes or in homework, but I guess that we need to practice writing Bash more.

(a) In Bash, we use $0, $1, $2 ... to get the command line arguments.

(b) The correct way to check this is to get the $PATH variable, replace the : with space, and loop this list and compare with the second parameter. This can be found in Notes 3. The way to write to $PATH variable is using export, which can be found in the Notes 1. Some minor mistakes:

- Some of us compare strings using eq, which is used for arithmetic comparison.
- Some of us does not translate the :. Without this step $PATH is not a list for traversal.
- Some of us use 'grep' to find whether the parameter is in the $PATH. Although this is a great idea, we can have false positives: e.g., $PATH = $/usr/bin$ while the second parameter is $/usr$.

(c) There are multiple ways to get the oldest regular file. We can either use commands or use the Bash processing to get it. Then we shall use ${#string} to get the filename length. One common mistake is that some of us find the youngest file, instead of the oldest one.

(d) There are multiple ways to solve this question. We can either use commands, or use Bash to traverse the files and find the files that are two-char long.

(e) We are supposed to write a valid Bash function. Please refer to notes about the format of a Bash function, how to call a function, and how to get the return (see Notes 3). We do not do well in this question; many of us use the C-style approach to either write a function or call the function.

(f) We can use $# to get the number of arguments, and use -d to check whether the argument is a valid directory.

(g) Some general mistakes:

- We should use backticks for command calling.
- We should use special syntax for arithmetic calculation.

(h) Note that we still get partial credit if we made any mistake mentioned above.

Question 4: Python Programming

[16 pts] In this question, you shall write a Python script to calculate the Euler's totient function of an integer n. Do not be scared away from the question; It is easy after you read and understand the following explanation. Hope that you also learn something from this question.

In number theory, Euler's totient function counts the positive integers up to a given integer n that are relatively prime (or co-prime)[1] to n, which is denoted as $\varphi(n)$. As an example, $\varphi(9) = 6$ since $1, 2, 4, 5, 7, 8$ are relatively prime to 9, while $3, 6, 9$ are not. $\varphi(10) = 4$ since $1, 3, 7, 9$ are relatively prime to 10, while $2, 4, 5, 6, 8, 10$ are not.

There exists an efficient way to calculate $\varphi(n)$. The critical step is to calculate prime factorization of a given n, which means that we will express an integer as the multiplication of a set of prime integers. For example, $360 = 2\times2\times2\times3\times3\times5 = 2^3\times3^2\times5^1$. After we get this factorization, it is easy to calculate the $\varphi(n)$. Suppose that $n = p_1^{k_1}\cdots p_r^{k_r}$, where $p_1\cdots p_r$ are different prime numbers, $\varphi(n) = n(1-1/p_1)(1-1/p_2)\cdots(1-1/p_r)$. For example, $10 = 2^1 \times 5^1$, and $\varphi(10) = 10 \times (1-1/2) \times (1-1/5) = 4$; $360 = 2^3 \times 3^2 \times 5^1$, and $\varphi(360) = 360 \times (1-1/2) \times (1-1/3) \times (1-1/5) = 96$.

Luckily enough, you are not asked to calculate the prime factorization of n. Instead, there exists an library called zhaowei that helps you do it. In this library, you can call a function called "p-factor", which takes an integer as the input, and outputs a dictionary which represents the prime factorization of the input.

Calls p-factor(10) returns a dictionary {2:1, 5:1}, since $10 = 2^1 \times 5^1$.
Calls p-factor(360) returns a dictionary {2:3, 3:2, 5:1}, since $360 = 2^3 \times 3^2 \times 5^1$.

Note that if the input for p-factor() is not an integer greater than 1, the output is an empty dictionary.

(a) Describe how to install the package zhaowei using Python package management tools.

(b) Write a Python 2 scripts that takes one parameter n from the standard input. Calculate and print the Euler's totient function $\varphi(n)$ of n. You shall utilize the library zhaowei and the function p-factor.

(c) Your script should output "error" if the input is not a positive integer, and $\varphi(n)$ otherwise. You **do not** have to consider any other exceptions. Hint: $\varphi(1) = 1$ should be considered separately because p-factor only works for integer greater than 1.

Average/Median/Max: 11.27/12.25/15.5
The outcome of this question is good. If you read through the description carefully, if is a short program within 20 lines of codes.

(a) Note that we are talking about Python package management here. The answer should either be pip or easy_install.

(b) We should pay attention to the following things

- We shall import the library zhaowei.
- We should correctly read the parameter from command line, e.g. using sys.argv. Please refer to our Python homework.
- We can iterate the returned dictionary by using something like 'for i in dict.keys()'.
- The calculation should be correct. Note that we are using the dictionary keys to do the calculation, not the values.

[1] Two integers x and y are said to be co-prime if the only positive integer that divides both of them is 1. E.g., 10 and 9 are co-prime, while 10 and 8 are not. As a side note, obviously, any two prime numbers are co-prime.

(c) This is a tricky question. The question asks us to print 'error' whenever the input is not a positive integer. However, most of us only consider when the input is is a non-positive integer. We should also consider the case where the input is non-integer rational numbers, or even irrational numbers (although they are not inputtable from standard input)! You can implement this by either directly judging whether the input is an integer, or you can utilize this feature of the function p-factor():

> "if the input for p-factor() is not an integer greater than 1, the output is an empty dictionary."

In other words, we can put an if statement after we get the return from p-factor(): whenever the returned dictionary is empty, we should output "error". Do not forget to handle the case where the input is 1 though.

(d) Note that we still get partial credit if we made any mistake mentioned above.

Question 5: C Programming: Multithreading, Git

[20 pts] There exists a Git directory in the remote Git server. `https://github.com/ZhaoweiTan/final.git`. This Git project only has one branch master. Inside this directory is a single file called vector.c. Inside the file it reads:

```
double dot_product(double v[], double u[], int n) {
    int i;
    double result = 0;
    for (i = 0; i < n; i+=1)
    {
      result += v[i]*u[i];
    }
    return result;
}
```

You are asked to do the following. Your answer should include the commands you use and the program you write.

(a) Git clone the code to your local computer under $HOME/test directory. (Suppose the directory is already there, but you have to change directory first)

(b) Create a new branch called "test" and switch to it.

(c) In this new branch, create a new file called mt_vector.c.

(d) Understand the code in vector.c; write a multithreading version of it in mt_vector.c. There should be at least a function called dot_product_multithread in your program, which utilizes **four (4)** threads and returns the identical results. When an error occurs when handling pthread, prints "error" and returns 999.

(e) Commit the changes and merge the changes to the master branch.

Average/Median/Max: 14.53/16.25/20
The outcome of this question is good. We are familiar with C programming, but we have to be more careful when writing the code.

Some general comments on git:

- All the commands that we are using can be found in the Slides. Note that for commit changes we should first 'add' then 'commit'. For merging, we should first switch back to the master branch and then merge the changes.

Some general comments on multithreading:

- We need to use the correct pthread function for creating and joining. Remember to check whether these functions return error whenever we use them.
- The parameter passing is crucial. We have to make sure the parameter is passed correctly from the main function to multiple threads. The safest way is to create multiple global variables, each of which store the subset of the initial vector that will later be processed by a thread.
- We need to add up the calculated value from multiple threads and collectively return the correct result.

Question 6: C Programming: System Call and Compiling

[20 pts]

(a) Firstly, please list the four compiling stages, and explain what each step roughly does.

(b) Suppose that you have a file called input.txt under your HOME directory. Write a main.c script. In the main function use system calls to read the content in this file.

(c) Suppose that the content in input.txt is a string. Write a C file called reverse.c and its header reverse.h. In these files write a function reverse_string that reverses a string.

(d) In main.c, call the function in reverse.c on the string you just read from input.txt. You should dynamically load reverse_string function in reverse.c using dlopen and dlsym. You are supposed to close up in the end as well.

(e) Use system call to write the reversed string to the standard output.

(f) Your program should consider all the linking errors and system call errors. If an error occurs, simply returns -1. Otherwise, a successful main should return 0.

(g) Write a simple makefile that compiles your library and program.

(h) What command can check the dynamic libraries your program uses?

Average/Median/Max: 15.28/16.75/20
The outcome of this question is good. We are familiar with C programming, but we have to be more careful when writing the code.

Some general comments on multithreading:

- We have to use the following system calls: open, read, write and close to open the file, read from that file, write to standard output, and close the file respectively. After each system call, we should check the return value in case there is error.
- For dynamic loading we should use dlopen and dlsym. To close up we use dlclose. Remember we need to check error after each of this.
- The Makefile should be working. Write something similar to Assignment 8 would be good enough.