

Name: _____

Write your name in the space provided above. Without looking at the test contents, write your initials on the top right corner of every *sheet* of paper. The last page is a reference.

Please wait to turn the page until everyone is told to begin.

While you are waiting, please read the following information:

There are 6 questions on 20 pages worth a total of 100 points. Please budget your time to get to all the questions. Keep answers brief and to the point.

Some question pages may be detached for your convenience. A stapler is available at the instructor podium if your entire exam falls apart.

The exam is closed book, closed notes, closed electronics, closed Internet, closed neighbor, closed telepathy, etc., with the exception of the references at the end of the packet.

Many of the questions have short solutions even if the question is somewhat long. Don't be alarmed.

If you don't remember the exact syntax of some command or the format of a command's output, make the best attempt you can. We will make allowances when grading. **Write legibly.**

Relax, you are here to learn.

Please wait to turn the page until everyone is told to begin.

CSE 374 Midterm Exam, 2/9/2015

Score: _____ / 100

1. _____ / 10

2. _____ / 16

3. _____ / 18

4. _____ / 18

5. _____ / 18

6. _____ / 20

Question 1. (10 points) (Shell commands) You've run a series of commands to explore your current directory. Note that `ls -F` just lists directories with a slash (/) at the end to distinguish them from regular files.

```
bash-$ pwd
/homes/admin
bash-$ ls -F
tools/          buildings.txt          maps/            buildings/
bash-$ ls -F maps
uw_south.jpg   uw_north.jpg         uw_west.jpg     uw_east.jpg
fields.jpg     ave.jpg
bash-$ cd tools
bash-$ ls -F
courses/      employees/
bash-$ ls -F courses
create_course.c   create_course.h   enroll_student.c
enroll_student.h
bash-$ ls -F employees
create_department.c   create_department.h   hire_employee.c
hire_employee.h
```

Solve each problem *individually*. The starting state of the shell is always the same.

a) You wish to refactor your programs so that header files are separate from source files. From the current shell state, write additional commands to create a new directory inside the `tools/` directory called `headers`. Move all `.h` files under the `tools` directory into the `headers/` directory. (at the end, `headers/` should only contain regular files not directories)

b) Each line in `buildings.txt` is the name of a building. You wish to start a new directory for each building. From the current shell state (i.e, ignore part (a) commands), write additional commands to create a new directory under `buildings/` for each line in `buildings.txt`.

CSE 374 Midterm Exam, 2/9/2015

c) You want a list of all UW map files. From the current shell state (i.e., ignore part (a,b) commands), write additional commands to create a new file `uw.txt` under `maps/` where each line is the filename in `maps/` that starts with `uw_`.

```
uw_south.jpg  
uw_north.jpg  
uw_west.jpg  
uw_east.jpg
```

CSE 374 Midterm Exam, 2/9/2015

Question 2. (16 points) (regular expressions) Give regular expressions that could be used with grep or egrep that would match the lines described. (You do not need to indicate grep or egrep, as long as your answer works with one or the other).

a) Lines containing a valid office number. A valid office number begins with CSE or EE, followed by an optional single space or underscore, followed by a three digit number. The first number is always between 1 and 6. Examples, CSE 100, CSE699, EE201, EE_222

b) Lines containing a pair of letters that occur consecutively at least 3 times. Examples, **insinuating**, **enlightenment**, **possessionlessness**

c) "i before e except after c". Lines containing ie (except for cie) or cei. For example, it should match **receive**, **sieve**, **thief**, BUT NOT match currencies.

d) Lines containing an odd number of characters.

Question 3. (18 points) (Shell scripting)

a) Consider the command that prints to stdout running processes:

```
bash-$ ps aux
USER      PID  %CPU %MEM  VSZ       RSS   TT  STAT  STARTED  TIME    COMMAND
alice    12277  0.0  0.1  2472836  6896   ??  S    Sat07PM  0:44.05 /usr/bin/top
alice    13275  0.0  0.0  2497564  3564   ??  S    Sat07PM  0:07.55 /usr/bin/display
root      274   0.0  0.1  2497520 10072   ??  Ss   Sat07PM  0:14.29 /usr/sbin/sshd
bob      10273  0.0  0.1  2490020  3321   ??  S    Sat07PM  0:03.11 /usr/bin/find
...
```

First column: username (who ran the program)

Second column: process id

Last column: the command that started the process

The whitespace between each column consists of one or more single spaces.

Give a one-line command (that includes `ps aux`) that prints to stdout the process ids of alice's processes. For example, given the above output of `ps aux`, your command would print out:

```
12277
13275
```

(HINT: `grep` and/or `sed` may be useful)

b) Write a shell script `stoplist` that kills all process ids provided in files. The command

```
stoplist f1 f2 ... fn
```

should kill process ids listed in each file. The format of the input files is one process id per line, with multiple lines.

To kill a process, use the `kill` command as follows:

```
kill -9 pid
```

- The `-9` argument is just required to forcibly stop the process.
- If `pid` is a valid process, then `kill` exits with a return code of 0 and prints nothing
- If `pid` was an invalid process, then `kill` exits with return code of 1 and prints an error message to `stderr`

Your script needs to follow this specification:

- print nothing to `stdout`
- print an informative error message to `stderr` for each invalid process id

CSE 374 Midterm Exam, 2/9/2015

- print an informative error message to stderr for each file argument that is not an existing regular file

You do not need to worry about permissions on files or processes.

Question 4. (18 points) (Shell scripts for text processing) Suppose that our shell's current working directory contains only directories, which are album names. For example:

```
bash-$ ls -F
Abbey Road/
Another Brick In The Wall/
The Dark Side of The Moon/
The Doors/
Please Please Me/
Thriller/
Are You Experienced/
...
Sgt Peppers Lonely Hearts Club Band/
```

Each directory contains a number of text files containing lyrics. They are numbered by track. For example:

```
bash-$ ls "Abbey Road"
01-Here Comes the Sun.txt
02-Because.txt
...
11-Her Majesty.txt
```

Solve each of the following problems individually (i.e. each problem starts with the same initial directories/files).

a) You decide that you don't want anyone else to see your music collection. Give a one-line command to turn off the group/others 'rwx' permissions for all albums and songs.

b) In one line, copy ALL of the song lyrics—substituting all occurrences of the word 'the' to the word 'a'—to a file called allLyrics.txt that is in your home directory.

c) Write a shell script that creates a file `songs.txt` in each album directory, which lists only the song *names*. For example, considering the example album, the script would create

“Abbey Road/songs.txt”

containing the contents

Here Comes the Sun

Because

...

Her Majesty

You may assume song names only consist of *alphanumeric characters and spaces*.

d) Write a script that prints to stdout the following: the album name, the number of words in each song, and the total number of words in the album. There is a blank line before the next album. Example of output shown (these are made-up word counts):

Abbey Road

100

222

201

111

425

1059

Thriller

300

201

233

344

1078

Question 5. (18 points) (Understanding C programs) Consider the following C program:

```
#include <stdio.h>
#include <string.h>

void mystery(int* a, int* b, int* c) {
    int* t = a;
    a = b;
    b = t;
    *c = *a - *b;
}

int main() {

    char foo[5] = "Hey";
    char bar[3] = "Yo";
    int u = strlen(foo);
    int x = 0;
    int* y = &x;
    int* z;
    int p = 12;

    strcpy(foo, bar);
    printf("%s %s\n", foo, bar);

    x = 1;
    z = &p;
    *y = *y + u;

    printf("XP: %d %d\n", x, p);
    printf("*: %d %d\n", *y, *z);
    mystery(z, y, &p);
    printf("XP: %d %d\n", x, p);
    printf("*: %d %d\n", *y, *z);
    z = y;
    mystery(z, y, &p);
    printf("XP: %d %d\n", x, p);
    printf("*: %d %d\n", *y, *z);
}
```

What output does this program produce when it is executed? (It is a valid C program and executes successfully). We suggest drawing diagrams showing variables and pointers to help you answer the question and help us award partial credit if needed. In your diagrams, if values change over time, cross out the old value and write the new value next to it rather than erasing the old value. (Hopefully, this will make it easier for us to follow your thought process).

Here is standard library documentation for your reference.

- `int strlen (char* str)`
 - returns the length of the string
- `void strcpy(char* destination, char* source)`
 - copy the string pointed to by source into the array pointed to by destination, including the null terminator.

Question 6. (20 points) (C Programming)

A strange electromagnetic pulse caused your office's computers to malfunction. There is one old vacuum-tube computer that still works and it has a C compiler! Since there is one computer to share, everyone is told to save all their needed calculations to a file that will be processed later. *It is your job to write a program that will process such a file and print out the answers.*

Each line of a file contains a left operand, followed by a single space, followed by + - * or /, followed by a single space, followed by a right operand.

For example, consider the file mylog.txt.

```
bash-$ cat mylog.txt
1 + 1
222 * 2
13 / 1
4 / 0
144 - 10
55 + 12
```

Assume the following about operands:

- between 1 and 3 digits long
- positive integer or 0

Here is what the program would do. Notice that if a line includes a divide-by-zero, "ERROR" is printed instead of a numeric result.

```
bash-$ ./calculate mylog.txt
2
444
13
ERROR
134
67
```

A coworker already has the code to read a file line-by-line. Your job is to implement the `calculate_line` function, which takes in *one line* of input, parses the input, does the calculation, and prints the required result line to stdout.

```
void calculate_line(char* line);
```

You should write auxiliary functions if and when they are appropriate. You don't need to write any `#include`'s; assume they are included for you.

Some (possibly useful) functions. There are multiple solutions. None should require all of these functions.

- `char* strncpy(dest, src, max_length)`
 - copy up to `max_length` characters from `src` to `dest`
- `int strlen(s, max_length)`
 - return the length of the string. If the string is longer than `max_length`, then `max_length` is returned.
- `int ctoi(char c)`
 - interpret the *character* `c` as an integer between 0 and 9. For example, `ctoi('0')` returns 0, `ctoi('1')` returns 1, ...
- `int stoi(char* s)`
 - interpret the *string* `s` as an integer. For example, `stoi("125")` returns 125.
- `int pow(int base, int exponent)`
 - return `baseexponent`. For example, `pow(3, 2)` returns 9.
- `int sscanf(char* input, char* format, ...)`
 - Reads data from `input` and stores them according to the parameter `format` into the locations pointed by the additional arguments (...). This is like `scanf` except it reads from a string instead of from `stdin`. Returns the number of items in the additional arguments that were successfully filled.

[this page intentionally blank]

Reference. **This is an incomplete list.**

- mkdir [options] directory_name ...
 - create a new directory for each argument. Exits 0 on success, and >0 if error occurs
 - options
 - -p create intermediate directories as required; also, no error reported if a directory already exists
- rm [options] file ...
 - remove the files
 - options
 - -r recursively remove the file hierarchy rooted at the file
 - -f if a file doesn't exist, don't display an error message

- wc [options] [file ...]
 - line, word, character count
 - example: (note the leading spaces)

```
wc foo.txt bar.txt
      1      1      5      foo.txt
      2      1      6      bar.txt
      3      2     11     totals
```

- only prints the totals row when there are 2 or more files
- options
 - -c print only characters and filename
 - -l print only lines and filename
 - -w print only words and filename

- for (shell)
 - example:

```
for i in x1 x2 x3 x4;
do
...
done
```

- if tests (shell)
 - ! expression true if expression is false
 - -d file true if file exists and is a *directory*
 - -e file true if file exists, regardless of type
 - -f file true if file exists, and is a *regular file*
 - -s file true if file exists and has a size greater than 0
 - -n string true if length of string is nonzero
 - -z string true if length of string is zero
 - string true if string is not the empty string
 - s1 = s2 true if the strings s1 and s2 are identical.

- `s1 != s2` true if the strings `s1` and `s2` are not identical.
- `n1 -eq n2` true if integer `n1` is equal to integer `n2`. Similar for `-ne -gt -ge -lt -le`
- `grep [options] pattern [file ...]`
 - options
 - `-n` line numbers
 - `-o` print only the matching part of the line (default is print whole matching line)
- `sed [options] command [file ...]`
 - options
 - `-e command` append the editing command to the list of commands
 - `-n`
 - By default, each line of input is echoed to the standard output after all of the commands have been applied to it. The `-n` option suppresses this behavior.
 - modifiers to regular expression: `s/.../.../[modifier]`
 - `p` print matched lines
 - `g` find multiple matches per line (default is first match)
- some regular expressions
 - `a+` one or more
 - `[abc$12]` one character in the list
 - `[^efg*45]` one character not in the list
 - `.` any single character
 - `^` beginning of line
 - `$` end of line
 - `(hello)` group
 - `\1 \2 \3` backreferences
- shell substitutions
 - ``cmd`` substitute with the stdout from running `cmd`
 - `$*` or `$@` all the arguments without quotes around them
 - `"$"` all the arguments are treated as one quoted string
 - `"$@"` all the arguments are treated as individual quoted strings
 - `$#` number of arguments; it is 0 for zero arguments