

IC221: Systems Programming
6-Week Written Exam
[SOLUTIONS]

February 12, 2014

Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, continue on the back of the page. Show all work, but please be legible.
Microscopic writing will not be graded. **This is a closed exam. No additional material may be used to complete this exam.**

Name: _____

Alpha: _____

| Question | Points | Score |
|----------|--------|-------|
| 1 | 15 | |
| 2 | 15 | |
| 3 | 15 | |
| 4 | 15 | |
| 5 | 15 | |
| Total: | 75 | |

1. For the following questions, assume a file system with the following structure. All directories end in /, all files have no set file extension, and ~ is the home directory.

```
~/
|
|--> foo/
|   |____.-> bar.txt
|   |____.-> foo
|
|--> bar/
|   |____.-> bar.jpg
|   |____.-> foo
|   |____.-> baz/
|
|---> baz/
```

- (a) [3 points] What is the output of running `ls` from the home directory?

```
foo bar baz
[+1 for each]
```

- (b) [3 points] What is the output of running the command `ls -a bar` from the home directory?

```
.. bar.jpg foo baz
[+2 for each named file, +1 for the . and ..]
```

- (c) [3 points] Write a glob pattern that will list, with `ls`, all files that start with `bar`. Assume that the `ls` is run from the home directory.

```
ls */bar.*
[+2 bar.* and +1 for having */]
```

- (d) [3 points] Write a `find` command that will find all directories and print them to the terminal. Assume that the `find` is run from the home directory.

```
find . -type d
[+3 for -type d, +1 for -type with no d]
```

- (e) [3 points] Write a `find` command with an `xargs` or `-exec` that will remove all files (*not* directories) named `foo`.

```
find . -type f -name foo -exec rm {} \;
OR
find . -type f -name foo | xargs rm
[+2 for -type f and -name foo, +1 for rm with exec or xargs]
```

2. For this question, assume you have a comma separate file (`database.csv`) with the following fields:

```
First Name,Last Name,Street Address,City,State,Phone Number
```

The state, is a two-character postal code (e.g. MD for Maryland and CA for California). The phone number is always formatted like 999-999-9999. The `database.csv` is not corrupted, and none of the fields may have a comma in it.

- (a) **[3 points]** Write a command line to count the number of unique states in `database.csv`. Your command may contain pipes and redirects.

```
cut -d "," -f 5 | sort | uniq | wc -l  
[+2 for good cut, +1 for sort, unqi and a wc]
```

- (b) **[3 points]** Write a command line to extract all the area codes of each of each of the phone numbers in `database.csv`. It is fine if the same area code is printed multiple times. Your command may contain pipes and redirects.

```
cut -d "," -f 6 | cut -d "-" -f 1  
[+2 for one cut, +1 for other cut]
```

- (c) **[3 points]** Write a command line that will save all lines in `database.csv` that contain the state code "MD", for Maryland, anywhere on a database line. Then save those lines to a file named `database.md.csv`. Your command may contain pipes and redirects.

```
grep "MD" database.csv > database.md.csv  
[+2 for either grep or redirect alone, +3 for both parts together, +1 for just writing grep]
```

- (d) **[3 points]** What are the three guiding principles of the Unix design philosophy?

```
(1) Write programs that do one thing and do that one thing well  
(2) Write program that work well together  
(3) Write programs to handle text streams  
[+1 for each principle]
```

- (e) **[3 points]** Using an answer from above, describe how that command meets the Unix design philosophy.

```
[+1 for using term pipeline, +1 for reefering to a prior command, and +1 for covering each of the principles.]
```

3. The questions below all reference this small program:

```

1  int main(){
2      int fd, n;
3      char buf[4096];
4
5      if( (fd = open("save.out", O_WRONLY | O_TRUNC | O_CREAT, 0644)) < 0){
6          fprintf(stderr, "ERROR: Cannot Open File\n");
7          return 2; //error!
8      }
9
10     while( (n = read(0, buf, 4096)) > 0 ){
11         if ( write(fd, buf, n) < 0 ){
12             fprintf(stderr, "ERROR: Bad Write\n");
13             return 2; //error!
14         }
15     }
16
17     if( n < 0 ){
18         fprintf(stderr, "ERROR: Invalid Read\n");
19         return 2;
20     }
21
22     close(fd);
23     return 0;
24 }

```

- (a) [4 points] In the code above, place a **circle** around all *library functions* and a **box** around all *system calls*. What is the purpose of a system call with respect to the user interaction with the operating system?

Library Functions: `fprintf()`
 System Calls: `open()`, `read()`, `write()`, `close()`
 A system call is a request to the O.S. to perform a privileged operation.
 [+2 for circling/boxing, +1 for explanation without "privileged operation", +2 for with]

- (b) [3 points] In the `open()` function call on Line 5, use the following option flags: `O_WRONLY`, `O_TRUNC`, `O_CREAT`. Describe in plain English what each of these flags mean and the method that is being used to combine them.

Open for writing, truncate if it exists, and create if it doesn't. Options are combined using an ORing.
 [+2 for explanation, +1 for ORing]

- (c) [4 points] What are the file descriptor numbers and names for each of the standard file descriptors? Which of those are being used in the above program?

0: Standard Input, 1: Standard Output, 2: Standard Error
 0, standard input, is used in the `read()`, `stderr` is used in `fprintf()`
 [+1 for each descriptor, +1 for which is used]

- (d) [4 points] What does the above program do?

Reads from standard input and saves results in file `save.out`, just like from lab 3. [+2 for reads from standard input, +2 for saves what's read to file `save.out`. Provide credit for other explanations of the open and error process, if given, but not full credit without the top 2.]

4. The questions below all reference the following bash script, `foobar_existence.sh`:

```

1  #!/bin/bash
2
3  for arg in $*
4  do
5      if [ $arg == "foo" ]
6      then
7          echo "You say foo, I say bar"
8      else
9          ls $arg >/dev/null 2>&1
10         res=$?
11
12         if [ $res -eq 0 ]
13         then
14             echo "$arg exists"
15         else
16             echo "$arg DOES NOT exist"
17         fi
18     fi
19 done

```

- (a) **[3 points]** On line 1, there is a special kind of comment to start the bash script. What does this comment do with respect to running the script in the terminal like, `./foobar_existence.sh`?

The shabang, or pound-bang, and it tells the terminal how to interpret the script, such as to treat this as bash.
[+1 for shabang or # bang, +2 interpret or treat as bash when executed]

- (b) **[3 points]** What values do the special bash variables `$*` and `$?` store? Refer to the script to explain.

`$*` stores the arguments, iterate through args in for loop, and `$?` stores result of operation, such as after the `ls`.
[+1 for each explanation, +1 for referring to the script in some way]

- (c) **[4 points]** On line 9, there is a sequence of redirects with the `ls` command. What standard file descriptors are redirected where?

Standard error gets redirect to standard out which all get's redirected to `/dev/null`
[+2 for stderr to stdout, +2 for stdout to /dev/null]

- (d) **[5 points]** Using the directory structure from Question 1, what is the output of running `foobar_existence.sh` from the home directory with the following arguments: `foo bar/foo xyzz bar/bar.jpg`.

You say foo, I say bar
bar/foo exists
xyzz DOES NOT exist
bar/bar.jpg exists
[+2 for "you say foo ...", +1 for each of the others]

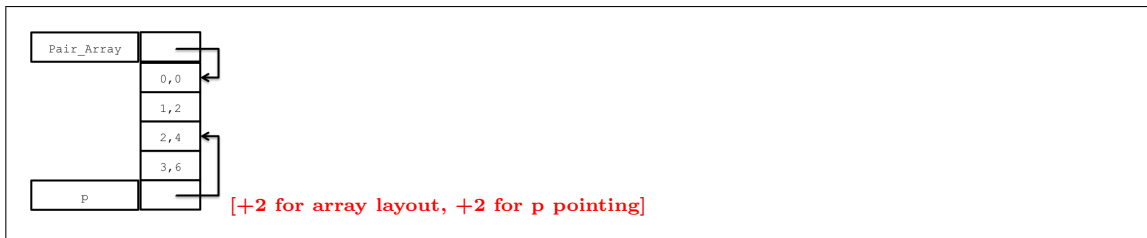
5. Consider the following C code:

```

1  typedef struct{
2     int left;
3     int right;
4  } pair_t;
5
6  int main(){
7     pair_t pair_array[4];
8     pair_t * p;
9     int i;
10
11    for( i=0; i<4; i++){
12        pair_array[i].left = i;
13        pair_array[i].right = i*2;
14    }
15
16    p = pair_array+2; /* MARK */
17
18    printf( "A: (%d,%d)\n", p->left, p->right);
19
20    printf( "B: (%d,%d)\n", p[1].left, p[1].right);
21
22    return 0;
23 }

```

(a) [4 points] Draw the stack/memory diagram for the above code at the MARK.



(b) [4 points] What is the output of this program?

A: (2,4)
 B: (3,6)
 [+2 for each]

(c) [4 points] What is the difference between the . and the -> operators when accessing structure members? Use the above code in your explanation.

The . operator is for when you working with the structure directory, and the -> is when you have a pointer to the structure, require a deference.
 [+2 for each description]

(d) [3 points] Does any memory need to be de-allocated in this program? If so, what? And, if not, why not?

No, everything is declared on the stack. No malloc().
 [+3 for No with explanation, +2 for No with no explanation, +1 for wrong answer with explanation]