2pts in total.

1. (1pt) Use the Unix file I/O system calls that we have discussed in the class to implement the Unix command "tee", which reads from the standard input (until reaching EOF) to each FILE, and also to the standard output.

The help page for the "tee" command is listed below. You can see it from your shell by ">$ tee --help".

You only need to handle the basic feature and support "--append". You do not need to implement the rest of the optional features ("OPTION").

```
Usage: tee [OPTION]... [FILE]...
Copy standard input to each FILE, and also to standard output.

  -a, --append              append to the given FILEs, do not 
overwrite
...
```

Example use case:
(you see file1 file2 file3 all contain the contents of /etc/passwd)

```
>$ cat /etc/passwd | tee file1 file2 file3
```

You do not have to submit compilable code. You only have to write pseudo code and explain why and how you use certain system calls. You don't need to handle parsing the arguments, options, etc. You are not required to do error handling.

You are asked to write up and explain why the code works to support the basic functionality of tee and the "--append" option.

If you submit the code without any explanation, you will get zero points.

2. (0.5 pt)

2.1 (0.4 pt)
The Bourne shell, Bourne-again shell, and Korn shell notation digit1>&digit2 says to redirect file descriptor digit1 to the same file as descriptor digit2. What is the difference between the two commands shown below? Specify what happens to the file descriptors and the open file table.

```
              ./a.out > outfile 2>&1
              ./a.out 2>&1 > outfile
```

(Hint: The shells process their command lines from left to right.)

2.2 (0.1 pt)
Describe an actual use case that you might want to do 2>&1 in your

shell.

3. Answer yes and no to the following questions. (0.5 pt in total, 0.1 pt each)

3.1. If you open a file for read—write with the append flag, can you still write from anywhere in the file using lseek?

3.2. In your Unix system, doing “cd ..” and “cd .” will always lead you to different directories.

3.3. When you type the command “ls” in your shell, a new process is created to run the “ls” program.

3.4. Consider the following code example we show in the class. Suppose func() is run by two processes in the system simultaneously. The reason why we want to make the seek() and write() here atomic is because one process could execute "seek()" after another process finishes executing "write()".

```
func() {
        ..
        seek(fd, 0, SEEK_END);
        write(fd, buf, 100);
        ..
}
```

3.5. The fcntl() function, if we use it correctly, provides the same functionality as dup() and dup2().