

EEE 3535 - Fall 2018

Project #1: Unix Utilities

Objectives:

To be familiar with the C programming language and Linux environment

Due Dates:

Softcopy: October 8th, 19:00PM (on YSCEC)

Hardcopy: October 8th, 19:00PM (Engineering Hall III C518)

** Late submission within 24 hours automatically deducts 30%.*

Things to Submit:

- ◆ Soft copy: *Name(Korean)_StudentID.zip*

Compress your source codes and executable files with your report

- ◆ Hard copy: *Report* only

Professor: W. W. Ro

T. A.: Jiwon Lee

Project 1 Unix Utilities

1. Install Linux Operating System on your Window environment

1.1 Download and install virtual machine

- Download and install virtual machine from one of the following webpages (**VirtualBox is preferred.**)

Virtual Box: <https://www.virtualbox.org/wiki/Downloads>

VMware Workstation Player: <https://www.vmware.com/kr/products/workstation-player.html>

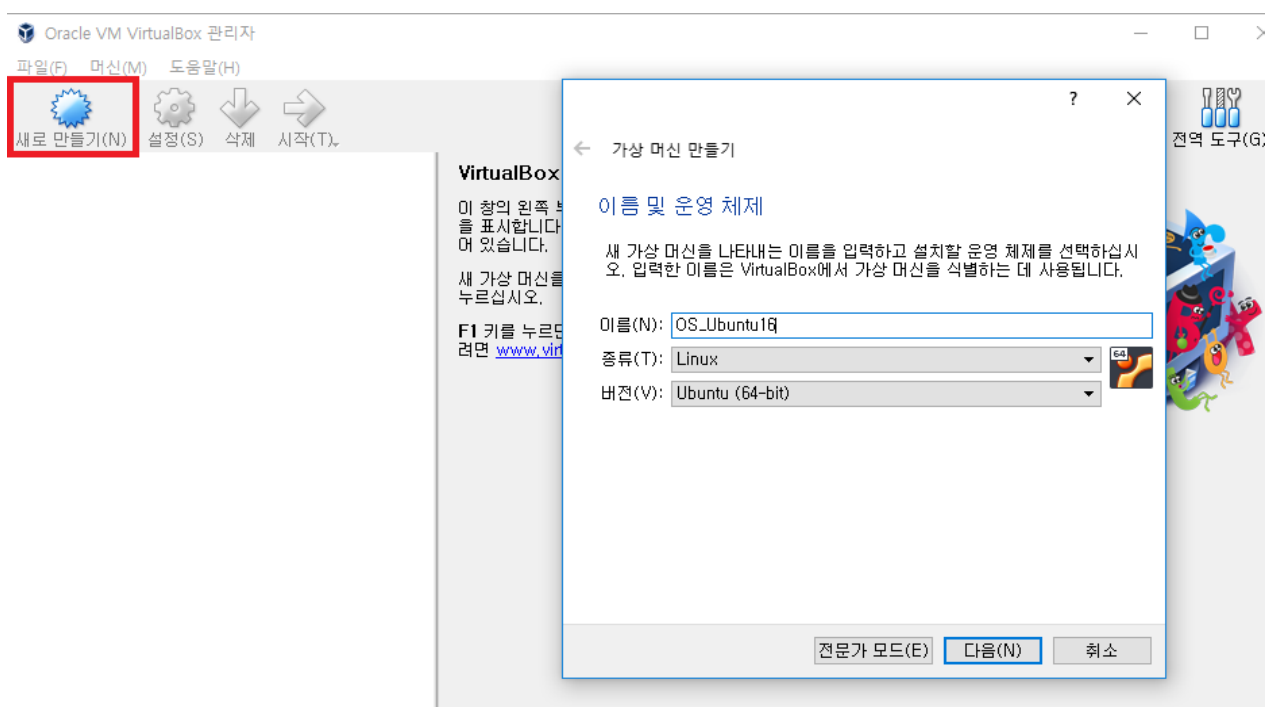
1.2 Download and install one of Linux Operating System on virtual machine

- You can download Ubuntu 16.04 LTS image file(.iso) from the following webpage
<http://releases.ubuntu.com/16.04/ubuntu-16.04.5-desktop-amd64.iso>

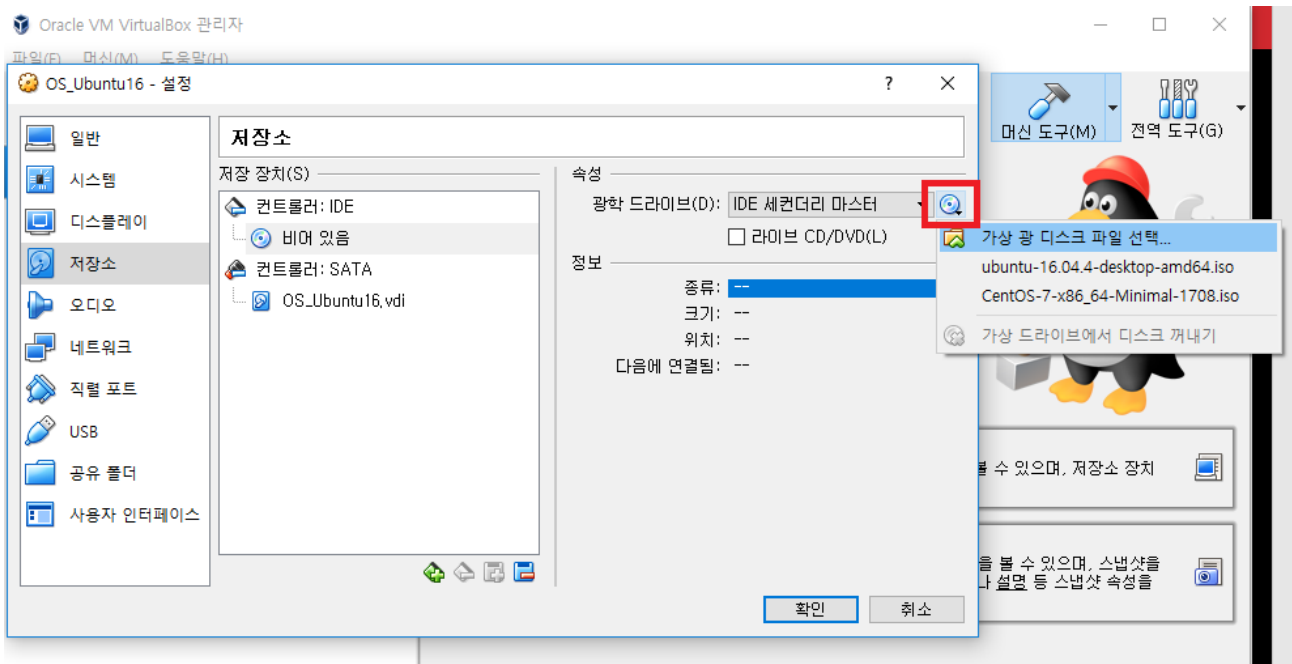
Others are not recommended.

- Install Ubuntu on virtual machine

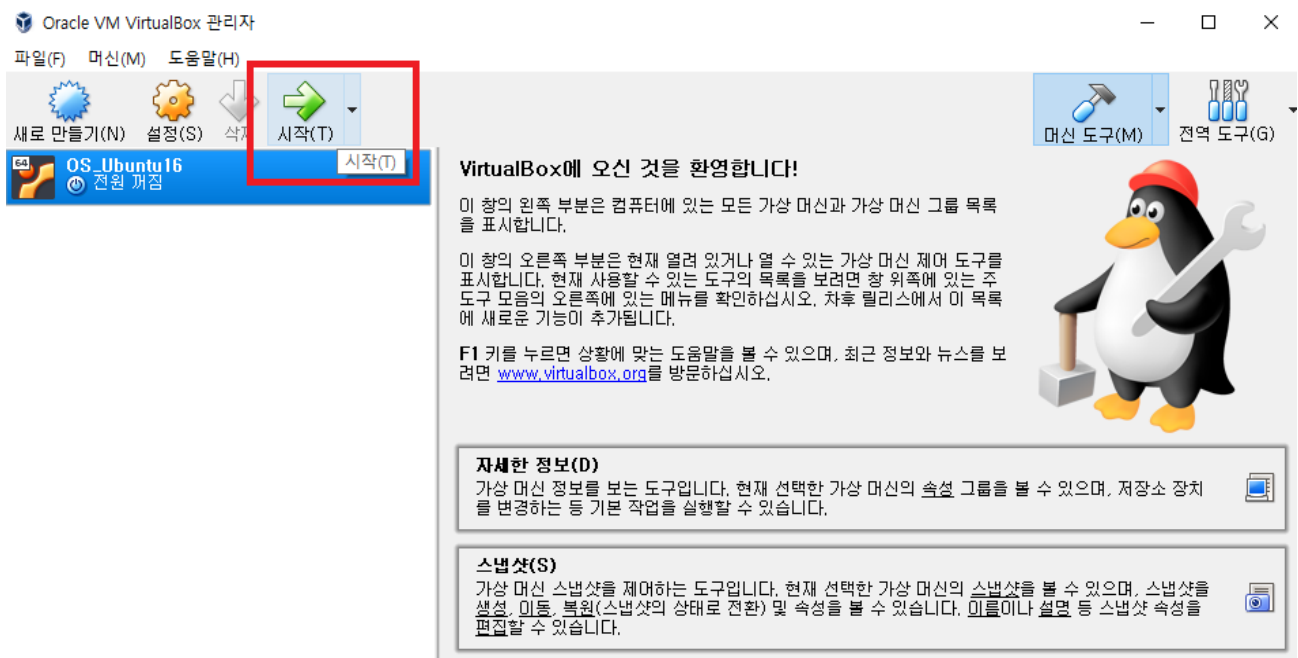
- ① Run the Virtual Machine
- ② Create a New Virtual Machine



③ Open the Linux image file(.iso)



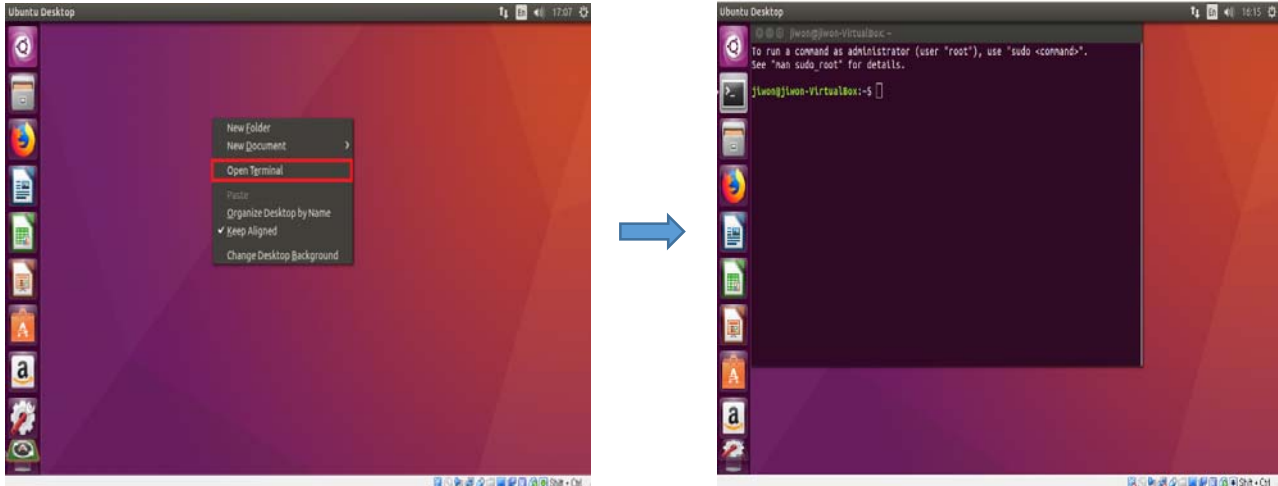
④ Start the Virtual Machine and Install Linux According to the Screen



※ Recommend you to configure the system in “English.”

2. How to Edit and Compile a Source Code

2.1 Run the terminal



2.2 Write your C code

- Generate source file “[Name of the Utility Program].c”.
- The name of the utility programs will be described in the section 3.
- If you enter the following command in terminal, you can write your C code.

\$ gedit [Name of the Utility Program].c (e.g., gedit my-cat.c)

```
jiwon@jiwon-VirtualBox: ~/Desktop/proj1
jiwon@jiwon-VirtualBox:~/Desktop/proj1$ gedit my-cat.c

*my-cat.c (~/Desktop/proj1) - gedit

#include <stdio.h>
#include <stdlib.h>
#define LINE_SIZE 100

int main(int argc, char *argv[]) {

    int i;
    FILE *fp[argc-1];
    char str_buf[LINE_SIZE];

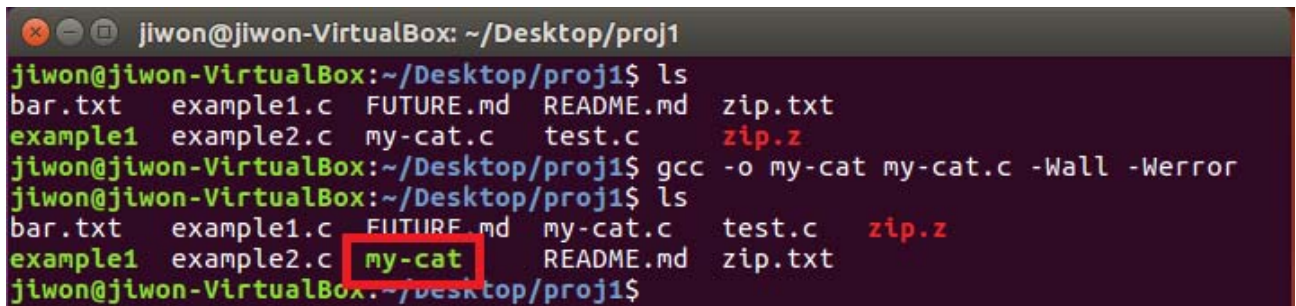
C Tab Width: 4 Ln 1, Col 1 INS
```

2.3 Compile your C code

- Compile the C code “[Name of the Utility Program].c” and generate executable file “[Name of the Utility Program]” using gcc.

\$ gcc -o [Name of the Utility Program] [Name of the Utility Program].c -Wall -Werror
(e.g., gedit -o my-cat my-cat.c -Wall -Werror)

- Your code should be compiled successfully when with the **-Wall** and **-Werror** flags.

A terminal window titled 'jiwon@jiwon-VirtualBox: ~/Desktop/proj1' showing the compilation of a C program. The user runs 'ls' showing files: bar.txt, example1.c, FUTURE.md, README.md, zip.txt, example1, example2.c, my-cat.c, test.c, and zip.z. Then they run 'gcc -o my-cat my-cat.c -Wall -Werror'. Another 'ls' command shows the same files, but 'my-cat' is highlighted with a red box, indicating the new executable file has been created.

```
jiwon@jiwon-VirtualBox: ~/Desktop/proj1
jiwon@jiwon-VirtualBox:~/Desktop/proj1$ ls
bar.txt  example1.c  FUTURE.md  README.md  zip.txt
example1 example2.c  my-cat.c   test.c     zip.z
jiwon@jiwon-VirtualBox:~/Desktop/proj1$ gcc -o my-cat my-cat.c -Wall -Werror
jiwon@jiwon-VirtualBox:~/Desktop/proj1$ ls
bar.txt  example1.c  FUTURE.md  my-cat.c  test.c  zip.z
example1 example2.c  my-cat     README.md zip.txt
jiwon@jiwon-VirtualBox:~/Desktop/proj1$
```

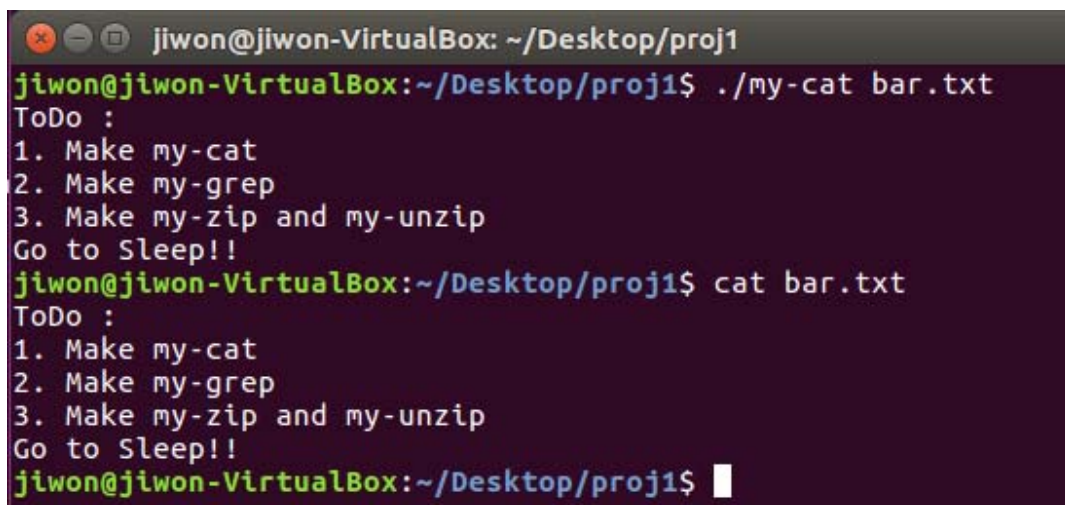
2.4 Execute your own program and compare results between your own programs and built-in Linux utilities to check functionality of your own program.

- To run your executable file, enter the following command.

\$./[your own program's name] (e.g., ./my-cat, ./my-zip)

- Built-in Linux utilities can be run on terminal with the name of the utility.

\$ [name of Unix utility] (e.g., cat, grep)

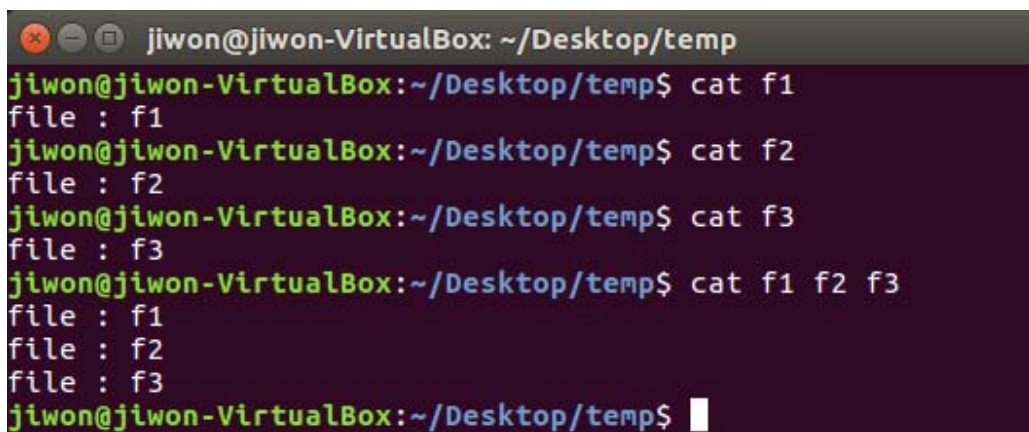
A terminal window titled 'jiwon@jiwon-VirtualBox: ~/Desktop/proj1' showing the execution of the 'my-cat' program and the built-in 'cat' utility. Both commands are used to read 'bar.txt', which contains a 'ToDo' list. The output of both commands is identical, confirming the functionality of the custom program.

```
jiwon@jiwon-VirtualBox: ~/Desktop/proj1
jiwon@jiwon-VirtualBox:~/Desktop/proj1$ ./my-cat bar.txt
ToDo :
1. Make my-cat
2. Make my-grep
3. Make my-zip and my-unzip
Go to Sleep!!
jiwon@jiwon-VirtualBox:~/Desktop/proj1$ cat bar.txt
ToDo :
1. Make my-cat
2. Make my-grep
3. Make my-zip and my-unzip
Go to Sleep!!
jiwon@jiwon-VirtualBox:~/Desktop/proj1$
```

3. Basic Unix Utility Implementation

3.1 my-cat

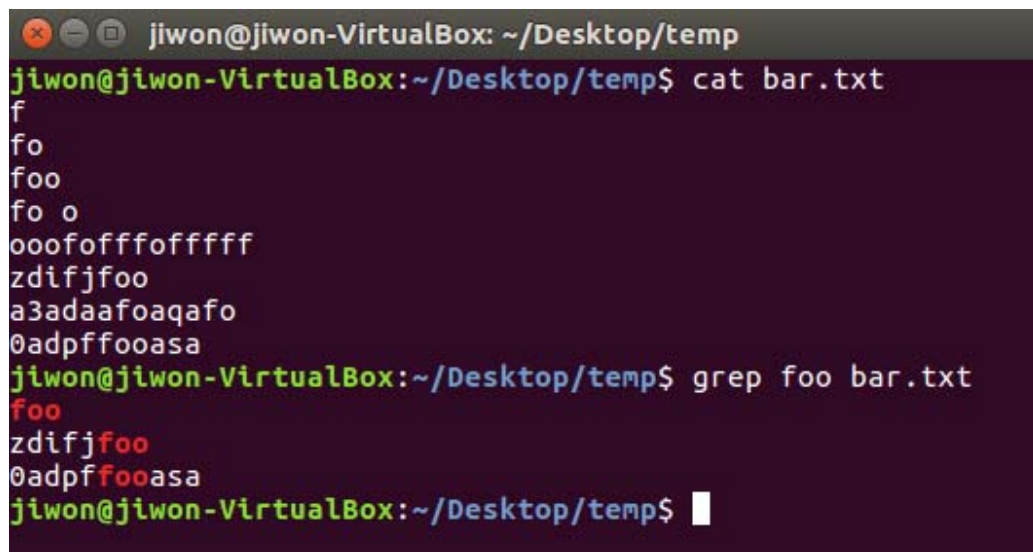
- my-cat is a simple program. It simply reads a file as specified by the user and prints its contents.
- You'll need to learn how to use a few library routines from the C standard library to implement the source code for this program. For my-cat, you might use **fopen()**, **fgets()**, **fclose()**, **printf()**.
- When you are using **fopen()**, you should check whether **fopen()** actually succeeded. You should always check if the call succeeded. If an error has occurred, you should print a message and then exit with error status of 1 using function **exit()**. You can also use the functions **perror()** or **strerror()** which are just optional.
- Note that you should *not* add a newline (`\n`) character to the **printf()**, because that would be changing the output of the file to have extra newlines. Just print the exact contents of the read-in buffer (which, of course, many include a newline).
- When you are done reading and printing, use **fclose()** to close the file to indicating you no longer need to read from it.
- Your program my-cat can be invoked with one or more files on the command line; it should just print out each file in turn.
- In all non-error cases, my-cat should exit with status code 0, usually by returning a 0 from **main()** or by calling **exit()**.
- If no files are specified on the command line, my-cat should just exit and return 0. Note that this is slightly different than the behavior of normal UNIX cat (if you'd like to, figure out the difference).
- If the program tries to **fopen()** a file and fails, it should print the exact message "my-cat: cannot open file" (followed by a newline) and exit with status code 1. If multiple files are specified on the command line, the files should be printed out in order until the end of the file list is reached or an error opening a file is reached (at which point the error message is printed and my-cat exits).



```
jiwon@jiwon-VirtualBox: ~/Desktop/temp
jiwon@jiwon-VirtualBox:~/Desktop/temp$ cat f1
file : f1
jiwon@jiwon-VirtualBox:~/Desktop/temp$ cat f2
file : f2
jiwon@jiwon-VirtualBox:~/Desktop/temp$ cat f3
file : f3
jiwon@jiwon-VirtualBox:~/Desktop/temp$ cat f1 f2 f3
file : f1
file : f2
file : f3
jiwon@jiwon-VirtualBox:~/Desktop/temp$
```

3.2 my-grep

- Your program **my-grep** is always passed a search term and zero or more files to grep through (thus, more than one is possible). It should go through each line and see if the search term is in it; if so, the line should be printed, and if not, the line should be skipped.
- The matching is case sensitive. Thus, if searching for **foo**, lines with **Foo** will *not* match.
- Lines can be arbitrarily long (that is, you may see many many characters before you encounter a newline character, `\n`). **my-grep** should work as expected even with very long lines. For this, you might want to look into the **getline()** library call (instead of **fgets()**), or roll your own.
- If **my-grep** is passed no command-line arguments, it should print "my-grep: searchterm [file ...]" (followed by a newline) and exit with status 1. Also, if **my-grep** encounters a file that it cannot open, it should print "my-grep: cannot open file" (followed by a newline) and exit with status 1. In all other cases, my-grep should exit with return code 0.
- If a search term, but no file, is specified, **my-grep** should work, but instead of reading from a file, **my-grep** should read from *standard input*. Doing so is easy, because the file stream **stdin** is already open; you can use **fgets()** (or similar routines) to read from it.
- For simplicity, if passed the empty string as a search string, **my-grep** can either match NO lines or match ALL lines, both are acceptable.



```
jiwon@jiwon-VirtualBox: ~/Desktop/temp
jiwon@jiwon-VirtualBox:~/Desktop/temp$ cat bar.txt
f
fo
foo
fo o
oooffofffofffff
zdifjfoo
a3adaafoaqafo
0adpffooasa
jiwon@jiwon-VirtualBox:~/Desktop/temp$ grep foo bar.txt
foo
zdifjfoo
0adpffooasa
jiwon@jiwon-VirtualBox:~/Desktop/temp$
```


4. Things to Submit

A. Unix Utility Executables and Source codes – soft copy only.

- A compressed file (zip, tar.gz, etc.) containing source codes and executables of your own work (total 2 source codes and 2 executables).

B. Report – both softcopy & hardcopy.

- Cover page with your name and Student ID.
- Screenshots for the result of running the 2 utility executable files.
- Write discussions on each of your screenshots.

5. Information

- You should make each utility within a single .c file.
- Each should compile successfully when compiled with the **-Wall** and **-Werror** flags. **-Wall** is a short for “warn all” which turns on (almost) all the warnings that gcc can tell you about. Using **-Werror** flags, all warnings are treated as errors.
- To read some functions in the terminal, try **man** on the terminal. For example, if you forgot how to use **fopen()** function, type “**man fopen**”. It will provide how a function is defined with some example.
- The full score of the project is 100 points. You should work alone. We recommend you read the website of Operating Systems: Three Easy Pieces project page as well, <https://github.com/remzi-arpacidusseau/ostep-projects/tree/master/initial-utilities>. Some ambiguous or uncertain explanation on this manual might be supplemented.