

12201922 이규민

## 2. Homework

1) Make a file with vi.

vi f1

(when you open a file with vi, sometimes you see Open, Edit, ... etc. at the bottom of the screen, in that case just type "e" in order to start editing)

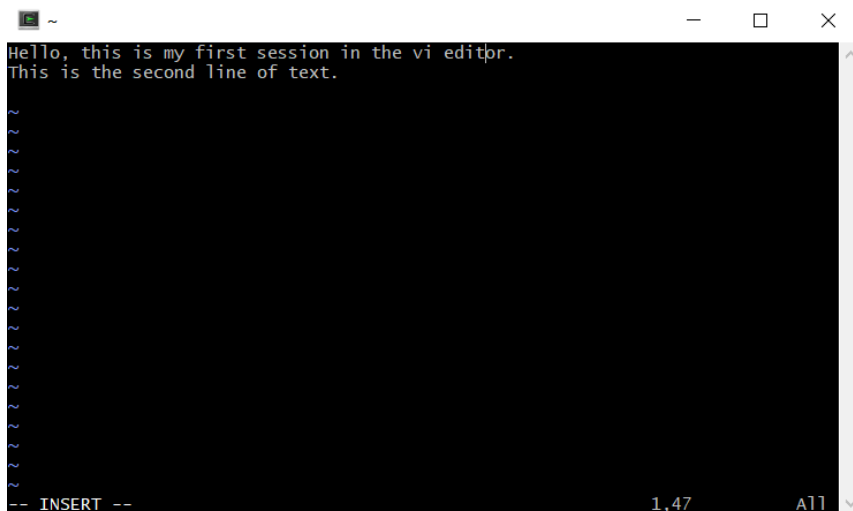


Vi f1 을 입력하면 command 창이 나타난다.

2) Start insertion with 'i' key and type following.

Hello, this is my first session in the vi editor.

This is the second line of text.

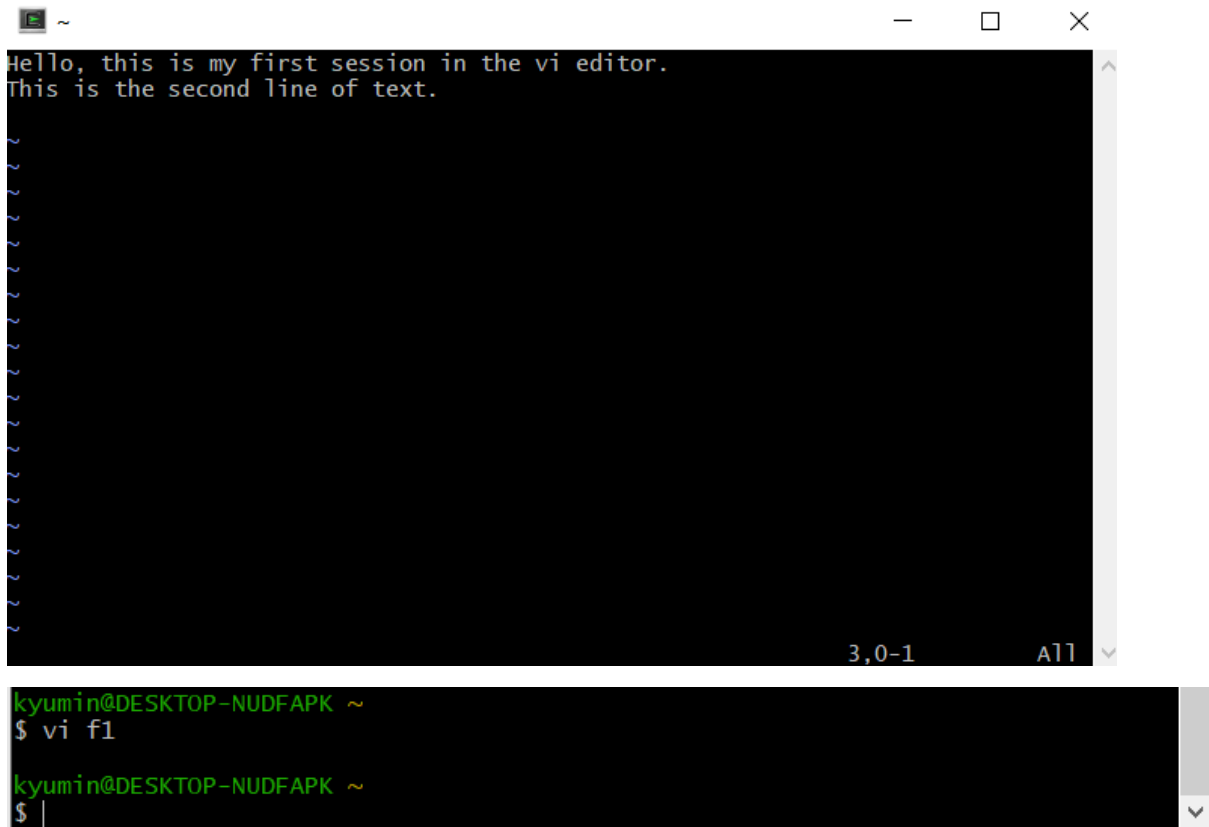


a, o, i 를 누르면 insert mode 로 진입할 수 있다. 이 경우 i 를 눌렀고, i 는 현재 커서에서 입력을 한다는 의미다.

3) Return to command mode with 'esc' key which is located at the top left position of your keyboard. Save and exit with ':wq'

esc

:wq

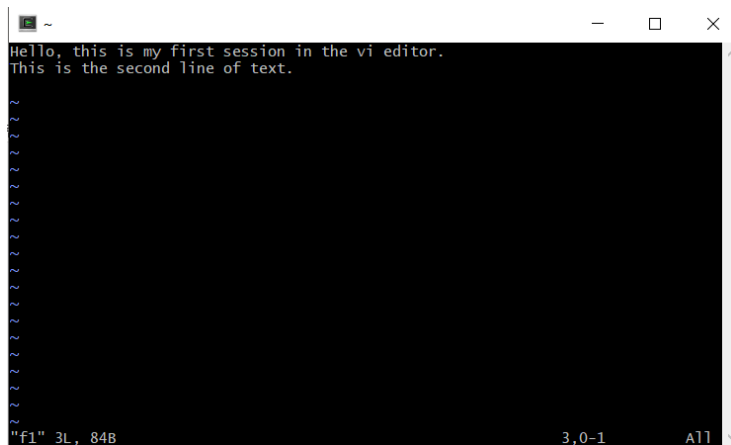


```
kyumin@DESKTOP-NUDFAPK ~  
$ vi f1  
kyumin@DESKTOP-NUDFAPK ~  
$ |
```

Esc 를 눌러 command mode 로 나온 후 :wq 를 입력하면 저장하고 나가기를 할 수 있다.

4) Reopen the file

vi f1



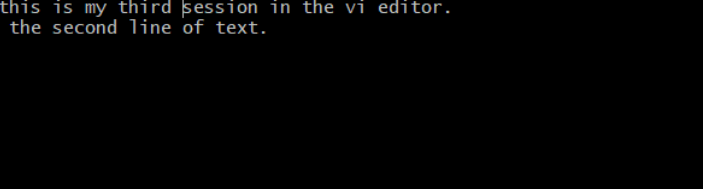
```
kyumin@DESKTOP-NUDFAPK ~  
$ vi f1  
kyumin@DESKTOP-NUDFAPK ~  
$ |
```

Vi f1 을 입력하면 아까 전의 command 창이 나타난다.

This is the second line of text.

A screenshot of a terminal window displaying the vi editor. The title bar at the top shows a file icon, a tilde (~), and standard window controls (minimize, maximize, close). The main area has a black background with white text. The first two lines are "Hello, this is my session in the vi editor." and "This is the second line of text.", with the cursor positioned at the end of the second line. Below these are several blank lines, each starting with a tilde (~) as a visual guide. At the bottom left, it says '"f1" 3L, 84B'. At the bottom right, there are two status indicators: "1,19" and "All". A vertical scrollbar is visible on the far right edge of the terminal window.

This is the second line of text.



A screenshot of a terminal window with a black background. The terminal shows the vi editor in insert mode. The text "Hello, this is my third session in the vi editor." is on the first line, and "This is the second line of text." is on the second line. The cursor is at the end of the second line. On the left side of the terminal, there are several tilde (~) characters. At the bottom of the terminal, the status bar displays "-- INSERT --", "1.25", and "All".

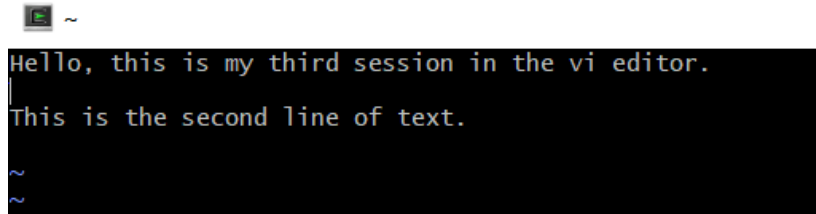
Insert mode 에서 third 라는 단어를 추가했다.

7) Add a new line as follows. Use ‘o’ key.

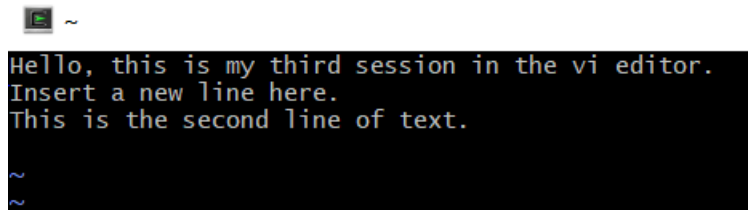
Hello, this is my third session in the vi editor.

Insert a new line here.

This is the second line of text.



```
~  
Hello, this is my third session in the vi editor.  
This is the second line of text.  
~  
~
```



```
~  
Hello, this is my third session in the vi editor.  
Insert a new line here.  
This is the second line of text.  
~  
~
```

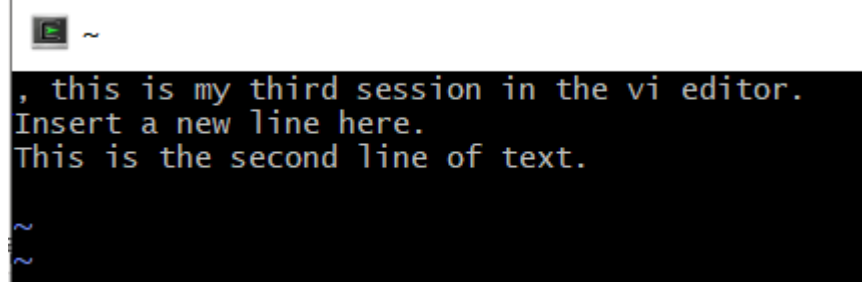
O 는 insert 모드로 진입하는 것이고, 커서를 새로운 라인에서 시작하라는 의미다. 첫째 줄에 커서를 잡고, o를 누르면 위 사진처럼 두번째 줄에 빈 줄이 생기는 것을 볼 수 있다.

8) Change the beginning as follows. Use ‘x’ and ‘i’ key.

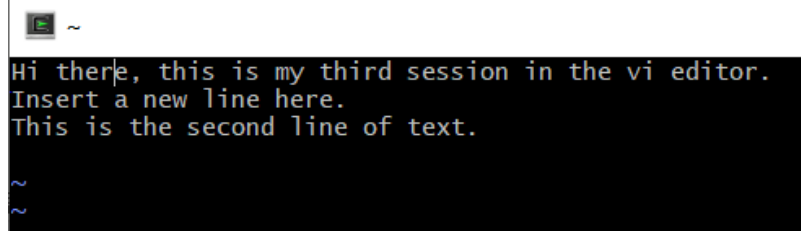
Hi there, this is my third session in the vi editor.

Insert a new line here.

This is the second line of text.



```
~  
, this is my third session in the vi editor.  
Insert a new line here.  
This is the second line of text.  
~  
~
```



```
~  
Hi there, this is my third session in the vi editor.  
Insert a new line here.  
This is the second line of text.  
~  
~
```

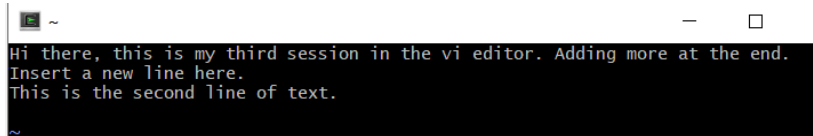
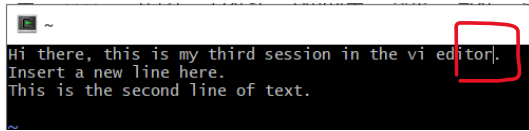
Command 모드에서 첫째 줄의 가장 왼쪽에 커서를 놓고 x를 눌러 글씨를 삭제하였다. I를 눌러 insert 모드에 진입 후 “Hi there” 라는 텍스트를 추가하였다.

9) Add more at the end. Use 'a' key.

Hi there, this is my third session in the vi editor. Adding more at the end.

Insert a new line here.

This is the second line of text.

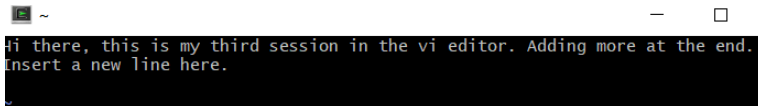


첫번째 사진처럼 커서를 잡고 i 를 누른다면 r 과 . 사이에서 입력이 되지만, a 를 누른다면 . 우측에서 입력이 된다. 즉 i 는 현재 커서에서 입력이 되고, a 는 현재 커서 다음 위치에서 입력이 된다는 것을 알 수 있다.

10) Delete the last line. Use 'dd'.

Hi there, this is my third session in the vi editor. Adding more at the end.

Insert a new line here.



Dd 를 누른다면 현재 커서가 있는 줄의 모든 내용을 제거한다. 사진에서는 세 번째 줄의 내용이 모두 제거된 것을 볼 수 있다.

11) Add few more lines. Use 'o'.

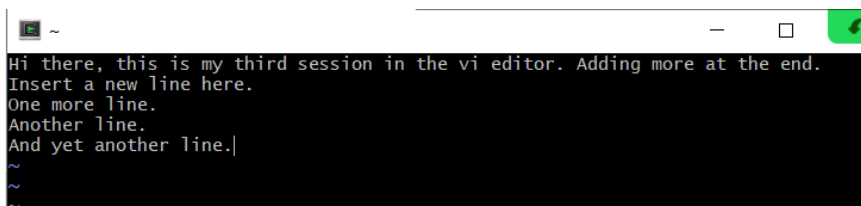
Hi there, this is my third session in the vi editor. Adding more at the end.

Insert a new line here.

One more line.

Another line.

And yet another line.



두 번째 줄에 커서를 잡고, insert 모드 진입 시 o 를 눌러 다음줄에 내용을 입력하였다.

12) Change the last line.

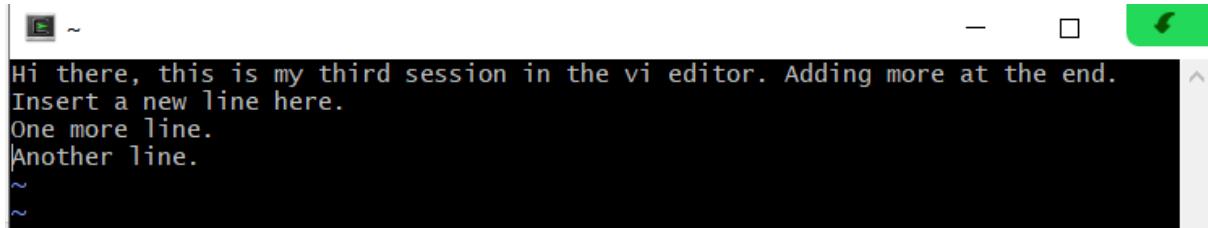
Hi there, this is my third session in the vi editor. Adding more at the end.

Insert a new line here.

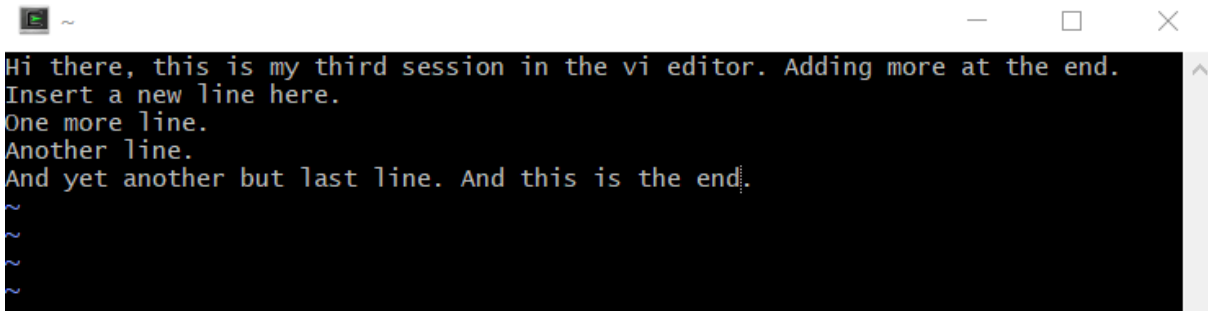
One more line.

Another line.

And yet another but last line. And this is the end.



```
~  
Hi there, this is my third session in the vi editor. Adding more at the end.  
Insert a new line here.  
One more line.  
Another line.  
~  
~
```



```
~  
Hi there, this is my third session in the vi editor. Adding more at the end.  
Insert a new line here.  
One more line.  
Another line.  
And yet another but last line. And this is the end.  
~  
~  
~  
~
```

Command 모드에서 5 번째 줄에 커서를 잡고 dd 를 눌러 5 행의 내용을 모두 제거했고, o 를 눌러 새로운 라인에 커서를 잡고, 내용을 입력하였다.

13) Copy and paste as follows. Use '2yy' to copy two lines at the current cursor; move the cursor to another location and use 'p' to paste them at that location.

Hi there, this is my third session in the vi editor. Adding more at the end.

One more line.

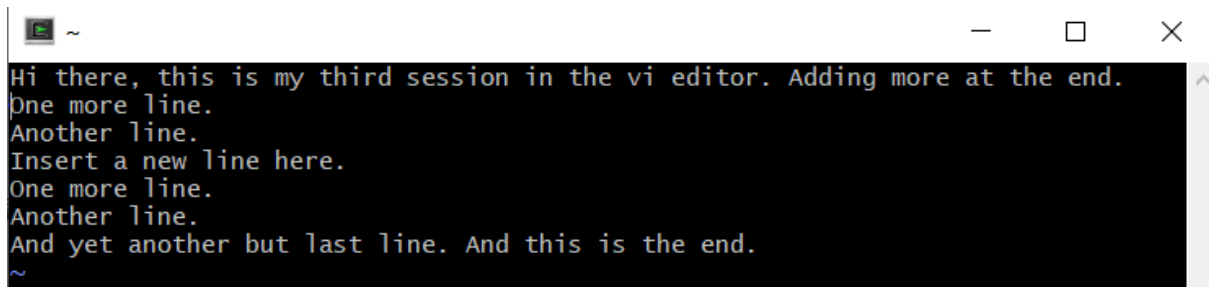
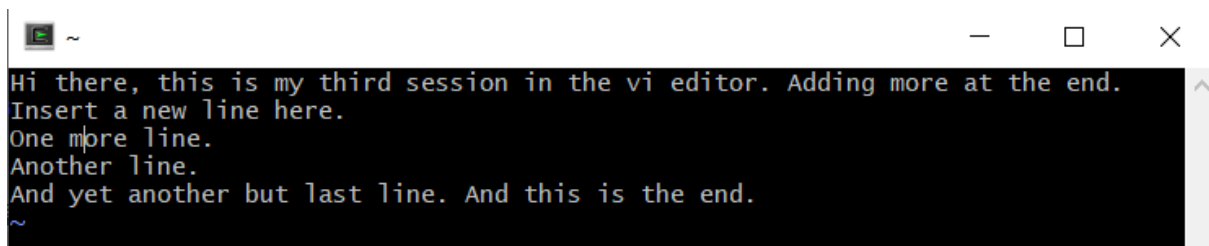
Another line.

Insert a new line here.

One more line.

Another line.

And yet another but last line. And this is the end.



세 번째 줄에 커서를 잡고, 2yy 를 입력하였다. 2yy 란 현재 위치한 줄을 포함하여 2 줄의 내용을 복사하라는 의미다. 이후 첫 번째 줄에 커서를 잡고 p 를 눌러 내용을 붙여넣기 하였다.

14) Go to line 6 with ‘:6’ and make change as follows.

Hi there, this is my third session in the vi editor. Adding more at the end.

One more line.

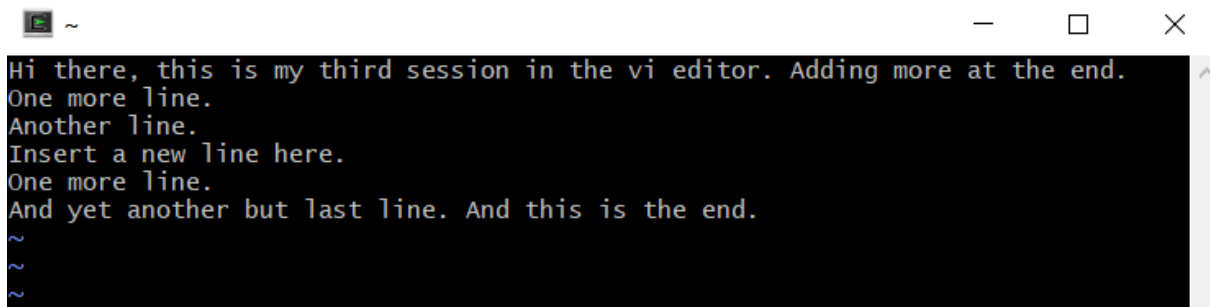
Another line.

Insert a new line here.

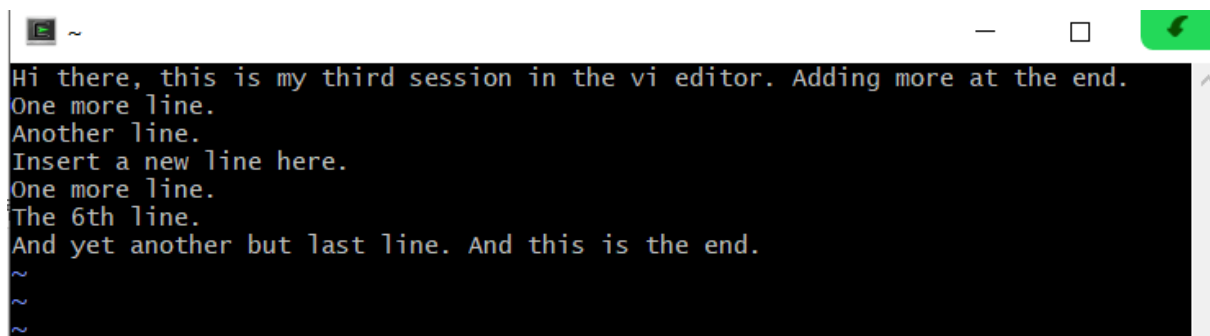
One more line.

The 6th line.

And yet another but last line. And this is the end.



```
~  
Hi there, this is my third session in the vi editor. Adding more at the end.  
One more line.  
Another line.  
Insert a new line here.  
One more line.  
And yet another but last line. And this is the end.  
~  
~  
~
```



```
~  
Hi there, this is my third session in the vi editor. Adding more at the end.  
One more line.  
Another line.  
Insert a new line here.  
One more line.  
The 6th line.  
And yet another but last line. And this is the end.  
~  
~  
~
```

:6 은 6 번째 줄에 커서를 잡으라는 의미다. Dd 를 눌러 6 번째 줄의 내용을 제거했고, 5 번째 줄에 커서를 잡은 후 o 를 눌러 새로운 라인을 추가했고, 그곳에 “The 6 th line.”을 추가했다.



15) Write a simple c program and compile and run.

Write a program:

```
$ vi ex1.c
```

```
#include <stdio.h>

void main(){
    printf("hi there\n");
}
```

Compile:

```
$ gcc -o ex1 ex1.c
```

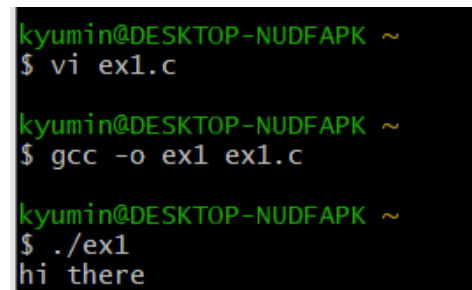
Run:

```
$ ./ex1
```

```
hi there
```

A screenshot of a terminal window with a dark background. The window title is '~'. The code displayed is: 

```
#include <stdio.h>
void main(){
    printf("hi there\n");
}
```

A screenshot of a terminal window with a dark background. The prompt is 'kyumin@DESKTOP-NUDFAPK ~'. The commands and output are: 

```
$ vi ex1.c
kyumin@DESKTOP-NUDFAPK ~
$ gcc -o ex1 ex1.c
kyumin@DESKTOP-NUDFAPK ~
$ ./ex1
hi there
```

Vi ex1.c로 command 모드 진입 후 c언어로 코드를 입력하였다. :wq로 저장하고 나온 후 gcc -o ex1 ex1.c를 입력하였다. 이것은 컴파일이며 ex1.c에 입력된 c언어를 ex1이라는 파일에 기계어로 저장하는 과정이다. ./ex1으로 해당 파일을 실행했고 정상적으로 "hi there"라는 메시지가 출력 되는 것을 볼 수 있다.

## 2. Homework

1) Login to the system. Show the current directory. Show what files are in your directory.

```
kyumin@DESKTOP-NUDFAPK ~  
$ pwd  
/cygdrive/c/Users/kyumin/AppData/Roaming/SPB_Data  
  
kyumin@DESKTOP-NUDFAPK ~  
$ ls  
cdssetup  d1  d2  f1
```

Pwd 를 통해 현 위치 확인이 가능하고, ls 를 이용해 현 위치의 파일들을 볼 수 있다.

2) Go to “/etc” directory. "file \*" will show the information for all files in the current directory. Combine "file \*" and "grep" using the pipe symbol(|) to display file information only for text files.

```
$ file * | grep text
```

```
kyumin@DESKTOP-NUDFAPK /etc  
$ file *  
DIR_COLORS:      ASCII text  
alternatives:    directory  
bash.bash_logout: ASCII text  
bash.bashrc:     ASCII text  
bash_completion.d: directory  
crypto-policies: directory  
defaults:        directory  
fstab:           ASCII text  
fstab.d:         sticky, directory  
hosts:           symbolic link to /  
  
kyumin@DESKTOP-NUDFAPK /etc  
$ file * | grep text  
DIR_COLORS:      ASCII text  
bash.bash_logout: ASCII text  
bash.bashrc:     ASCII text  
fstab:           ASCII text  
man_db.conf:     ASCII text  
nsswitch.conf:   ASCII text  
profile:         ASCII text  
shells:          ASCII text  
vimrc:           ASCII text, with escape sequences  
xattr.conf:      ASCII text
```

File \*은 모든 파일의 타입을 출력하는 명령어고, |파이프는 앞의 명령어의 실행결과를 뒤의 명령어의 입력으로 넘길 수 있고, grep text 를 입력하면 text 형식의 파일들만 찾아서 출력하라는 명령어다.

위 사진처럼 Etc 디렉토리 안에는 매우 많은 파일이 있다. 여기서 file \* | grep text 명령어를 사용한다면 text 파일들만 확인할 수 있다.

3) (If your Cygwin has no /etc/passwd, make one with “mkpasswd > /etc/passwd”.)  
Find the location of the password file (“passwd”), the location of C header files such as “stdio.h”, and the location of utility programs (or Linux commands) such as “ls”. Use “whereis” command. What is the difference between /usr/bin/passwd and /etc/passwd? Use “ls -l /usr/bin/passwd /etc/passwd” and “file /usr/bin/passwd /etc/passwd” to explain the difference.

```
kyumin@DESKTOP-NUDFAPK /etc
$ mkpasswd > /etc/passwd
```

/etc 에 Passwd 파일이 없기 때문에 mkpasswd > /etc/passwd 명령어로 파일을 만들었다.

```
kyumin@DESKTOP-NUDFAPK /
$ whereis passwd
passwd: /usr/bin/passwd.exe /etc/passwd /usr/share/man/man1/passwd.1oss1.gz

kyumin@DESKTOP-NUDFAPK /
$ whereis stdio.h
stdio: /usr/include/stdio.h

kyumin@DESKTOP-NUDFAPK /
$ whereis ls
ls: /usr/bin/ls.exe /usr/share/man/man1/ls.1.gz
```

Whereis 명령어로 passwd, stdio.h, ls 의 실행파일 및 man 페이지 위치를 확인하였다.

```
kyumin@DESKTOP-NUDFAPK /
$ ls -l /usr/bin/passwd.exe
-rwxr-xr-x 1 kyumin Administrators 22547 Feb 27 21:02 /usr/bin/passwd.exe

kyumin@DESKTOP-NUDFAPK /
$ ls -l /etc/passwd
-rw-r--r-- 1 kyumin 없음 1121 Mar 13 09:13 /etc/passwd
```

Ls -l 명령어를 사용하여 /usr/bin/passwd 와 /etc/passwd 의 차이점을 확인했다. Passwd.exe 는 다른 그룹원이나 다른 사람들도 실행할 수 있는 권한이 있다. 하지만 passwd 는 오직 오너만 실행할 수 있고, 그 외의 사람들은 읽기 권한밖에 없다.

```
kyumin@DESKTOP-NUDFAPK /
$ file /usr/bin/passwd.exe
/usr/bin/passwd.exe: PE32+ executable (console) x86-64, for MS Windows, 11 sections

kyumin@DESKTOP-NUDFAPK /
$ file /etc/passwd
/etc/passwd: CSV text
```

File 명령어로 /usr/bin/passwd 와 /etc/passwd 의 차이점을 확인했다. Passwd 는 텍스트 파일이지만 passwd.exe 는 실행파일이다.

4) Go to your login directory ("cd" without arguments will move you to your login directory).  
Make ex1.c using vi. Compile with "gcc" and run.  
vi ex1.c

```
#include <stdio.h>

void main(){

    printf("hello\n");

}
```

```
gcc -o ex1 ex1.c

./ex1

hello
```

To compile with g++ , change void main() => int main()

```
#include <stdio.h>

int main(){

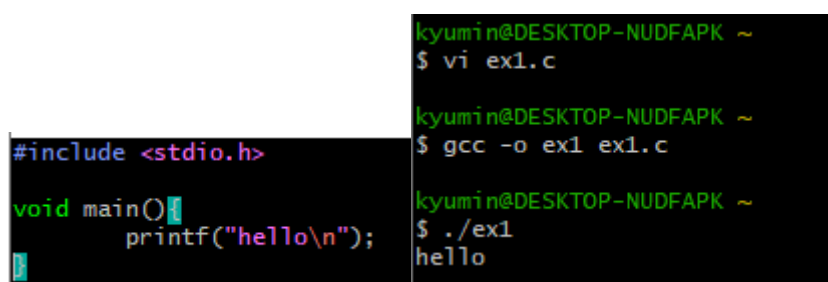
    printf("hello\n");

}

g++ -o ex1 ex1.c

./ex1

hello
```

A screenshot of a terminal window showing the process of creating and running a C program. The left pane shows the contents of the file ex1.c as edited in the vi editor, with syntax highlighting for C code. The right pane shows the terminal output, including the command to view the file, compile it with gcc, and execute it, resulting in the output 'hello'.

```
kyumin@DESKTOP-NUDFAPK ~  
$ vi ex1.c  
  
kyumin@DESKTOP-NUDFAPK ~  
$ gcc -o ex1 ex1.c  
  
kyumin@DESKTOP-NUDFAPK ~  
$ ./ex1  
hello
```

Vi ex1.c 명령어로 코맨드 모드 - 인서트 모드 진입 후 c 언어로 코드를 작성하였다. 이후 gcc -o ex1 ex1.c 명령어로 ex1.c 를 ex1 으로 컴파일했다. 이후 ./ex1 명령어로 컴파일 된 프로그램을 실행했고, 정상적으로 메시지가 출력된다.

```
kyumin@DESKTOP-NUDFAPK ~  
$ vi ex1.c  
  
kyumin@DESKTOP-NUDFAPK ~  
$ g++ -o ex1 ex1.c  
  
kyumin@DESKTOP-NUDFAPK ~  
$ ./ex1  
hello
```

Vi ex1.c 명령어로 코맨드 모드 - 인서트 모드 진입 후 c++ 언어로 코드를 작성하였다. 이후 g++ -o ex1 ex1.c 명령어로 ex1.c 를 ex1 으로 컴파일했다. 이후 ./ex1 명령어로 컴파일 된 프로그램을 실행했고, 정상적으로 메시지가 출력된다.

5) Display the contents of ex1.c using cat and xxd. With xxd, you can see the ascii code for each character in ex1.c. Find the ascii codes for the first line of the program: "#include <stdio.h>".

```
kyumin@DESKTOP-NUDFAPK ~  
$ cat ex1.c  
#include <stdio.h>  
  
int main(){  
    printf("hello\n");  
}  
  
kyumin@DESKTOP-NUDFAPK ~  
$ xxd ex1.c  
00000000: 2369 6e63 6c75 6465 203c 7374 6469 6f2e  #include <stdio.  
00000010: 683e 0a0a 696e 7420 6d61 696e 2829 7b0a  h>..int main(){.  
00000020: 0970 7269 6e74 6628 2268 656c 6c6f 5c6e  .printf("hello\n  
00000030: 2229 3b0a 7d0a                                     ");;}.  
$
```

Cat ex1.c 는 이전에 c++ 언어로 작성했던 코드 내용을 볼 수 있다. Xxd ex1.c 는 작성했던 문자들을 16 진수로 확인이 가능하다. 예를 들면 위 사진에서 초록색으로 2369 가 확인이 된다. 23 과 69 가 16 진수이고, 그 숫자를 아스키 코드표를 통해서 확인해보면 문자로는 각각 “#”과 “i”임을 알 수 있다.

6) Display the contents of ex1 (the executable file). You cannot use "cat" to see ex1. Why?

```
kyumin@DESKTOP-NUDFAPK ~  
$ cat ex1  
MZ#####!L!This program cannot be run in DOS mode.  
$PEd#####&  
#####P#####  
#####.text''.data' @.rdata@@.buildid5@@.pdata=P@@.xd  
atad'@@.bss=p@@.idata'@@.rsrc'@@.reloc  
0@B/45@#####"AB/40$@B/19k1@B/314  
kff. D#####UH H 3H 1H 4 H 7D#####% 1 H  
p#####%p#####%Bp#####H (1 U o#####D#####D  
h H H 5j H H ,^H 58H H H H 5f H H (H H H 4H ^H H
```

cat ex1 을하면 이상한 문자들이 출력된다. 그 이유는 ex1 파일은 ex1.c 가 기계어로 변환된 파일이기 때문이다.

6-0) You can look at the machine code corresponding to the C code in ex1.c with "objdump".

```
$ objdump -D -M intel ex1 > x
```

\$ vi x

Search for <main>.



7) Copy ex1.c to ex2.c, ex3.c, and ex4.c. Remove ex2.c. Rename ex3.c to y.c.

```
kyumin@DESKTOP-NUDFAPK ~
$ cp ex1.c ex2.c

kyumin@DESKTOP-NUDFAPK ~
$ cp ex1.c ex3.c

kyumin@DESKTOP-NUDFAPK ~
$ cp ex1.c ex4.c

kyumin@DESKTOP-NUDFAPK ~
$ rm ex2.c

kyumin@DESKTOP-NUDFAPK ~
$ mv ex3.c y.c

kyumin@DESKTOP-NUDFAPK ~
$ ls
cdssetup d1 d2 ex1.c ex1.exe ex4.c f1 x y.c
```

```
kyumin@DESKTOP-NUDFAPK ~
$ cat ex4.c
#include <stdio.h>

int main(){
    printf("hello\n");
}

kyumin@DESKTOP-NUDFAPK ~
$ cat y.c
#include <stdio.h>

int main(){
    printf("hello\n");
}
```

Cp 명령어를 사용하여 ex1.c 파일을 복사했다. Ex2, ex3, ex4 가 복사된 파일이며, rm 명령어로 ex2.c 파일을 제거했다. Mv 명령어로 ex3.c 파일의 이름을 y.c 로 변경했다. 파일을 복사하고, 이름을 바꿔도 파일 내용은 동일한 것을 알 수 있다.

8) Make a subdirectory. Copy y.c in this subdirectory.

```
kyumin@DESKTOP-NUDFAPK ~
$ ls
cdssetup d1 d2 ex1.c ex1.exe ex4.c f1 x y.c

kyumin@DESKTOP-NUDFAPK ~
$ mkdir exdir

kyumin@DESKTOP-NUDFAPK ~
$ cp y.c exdir

kyumin@DESKTOP-NUDFAPK ~
$ cd exdir

kyumin@DESKTOP-NUDFAPK ~/exdir
$ ls
y.c
```

Mkdir 명령어로 디렉토리를 만들었고 cp 명령어를 사용하여 하위 디렉토리 안에 y.c 파일을 복사하였다.

9) Redirect the output of ex1 to another file using ">" symbol and check its content with cat.

```
$ ./ex1 > f1
$ cat f1

kyumin@DESKTOP-NUDFAPK ~
$ ./ex1 > f1

kyumin@DESKTOP-NUDFAPK ~
$ cat f1
hello
```

./ex1 > f1 은 ex1 실행프로그램에서 출력 되는 “hello” 메시지를 f1 파일의 내용으로 저장하라는 명령어다. F1 에 다른 내용이 저장 되어있었지만 위 사진과 같이 “hello” 메시지가 출력되는 것을 볼 수 있다.

10) Use grep to search "hello" in all files (use -nr option).

```
kyumin@DESKTOP-NUDFAPK ~
$ grep -nr "hello" *
d1/f3:1:hello
ex1.c:4:      printf("hello\n");
grep: ex1.exe: binary file matches
ex4.c:4:      printf("hello\n");
exdir/y.c:4:   printf("hello\n");
f1:1:hello
x:257:00001000: 6865 6c6c 6f00 0000 0000 0000 0000 0000  hello.....
y.c:4:   printf("hello\n");
```

Grep 명령어를 통해 “hello” 메시지가 들어간 파일의 위치를 검색했다. 이전 문제에서 만들었던 f1 파일의 내용이 “hello”로 수정되었는데 그 내용도 확인이 된다.

11) Find out what processes exist in your system. Use "ps -ef".

```
kyumin@DESKTOP-NUDFAPK ~
$ ps -ef
```

	UID	PID	PPID	TTY	STIME	COMMAND
	kyumin	761	1	?	14:57:40	/usr/bin/mintty
	kyumin	769	762	pty0	15:00:04	/usr/bin/ps
	kyumin	762	761	pty0	14:57:40	/usr/bin/bash

Ps -ef 를 이용하여 실행 중인 모든 프로세스를 자세하게 확인할 수 있다.

12) "ps -ef" shows all the processes in the Linux system ("ps -W" to see all processes including ones from Windows). How do you know which ones are running in the current terminal? Use "tty" for this purpose. Note that when a user logs in, the system allocates a terminal, and you can find the terminal number with "tty" command. What is your terminal number?

```
kyumin@DESKTOP-NUDFAPK ~
$ ps -W
```

	PID	PPID	PGID	WINPID	TTY	UID	STIME	COMMAND
	65540	0	0	4	?	0	10:22:37	System
	65660	0	0	124	?	0	10:22:37	Registry
	66036	0	0	500	?	0	10:22:37	C:\Windows\System32\smss.exe
	66328	0	0	792	?	0	10:22:37	C:\Windows\System32\csrss.exe
	66420	0	0	884	?	0	10:22:37	C:\Windows\System32\wininit.exe
	66492	0	0	956	?	0	10:22:37	C:\Windows\System32\services.exe
	(x86)\Google\GoogleUpdater\124.0.6342.2\updater.exe							
	82640	0	0	17104	?	0	10:22:37	C:\Program Files\Google\GoogleUpdater\124.0.6342.2\updater.exe
	(x86)\Google\GoogleUpdater\124.0.6342.2\updater.exe							
	71288	0	0	5752	?	0	10:22:37	C:\Program Files\Google\GoogleUpdater\124.0.6342.2\updater.exe
	(x86)\Google\GoogleUpdater\124.0.6342.2\updater.exe							
	775	762	775	18556	pty0	197609	15:05:38	/usr/bin/ps

```
kyumin@DESKTOP-NUDFAPK ~
$ tty
/dev/pty0
```

Ps -W 명령어를 사용하여 윈도우의 프로세스를 포함하여 모든 프로세스 확인이 가능하다. 또한 tty 항목의 마지막 줄에 ‘pty0’라고 적혀있으며, tty 명령어를 사용해보면 동일하게 ‘pty0’라고 확인이 된다. 0 번 터미널인 것을 알 수 있다.



13) Modify ex1.c so that it receives two numbers from the user and prints the sum. Use scanf() for this.

```
#include<stdio.h>

void main() {
    int a;
    int b;
    int r;

    scanf("%d", &a);
    scanf("%d", &b);
    r = a + b;
    printf("%d", r);
}
```

```
kyumin@DESKTOP-NUDFAPK ~
$ vi ex1.c

kyumin@DESKTOP-NUDFAPK ~
$ gcc -o ex2 ex1.c
```

```
kyumin@DESKTOP-NUDFAPK ~
$ ./ex2
78 45
123
```

Ex1.c 의 코드는 a,b 라는 수를 입력 받아서 더한 값을 출력하는 코드다. Ex2 파일로 컴파일했고, 프로그램 실행 시 정상적으로 덧셈이 되는 것을 확인했다.

14) Modify ex1.c so that it contains an infinite loop after printing "hello".

```
.....
printf("hello");
fflush(stdout); // to make it print hello immediately
for(;;);
.....
```

```
#include<stdio.h>

void main() {
    printf("hello");
    fflush(stdout);
    for(;;);
}
```

```
kyumin@DESKTOP-NUDFAPK ~
$ vi ex1.c

kyumin@DESKTOP-NUDFAPK ~
$ gcc -o ex2 ex1.c

kyumin@DESKTOP-NUDFAPK ~
$ ./ex2
hello
```

Hello 출력 후 무한 루프에 빠지는 함수를 만들었다. For 문 안에는 아무 내용도 없다. 프로그램을 실행해보면 hello 출력 후 무한 루프에 빠져 ctrl+C 단축키로 강제 종료해주었다.

15) Run the program with & at the end, and use ps to check its status. "&" puts the process in the background so that you can type next command.

\$ ./ex1 &

\$ ps

```
kyumin@DESKTOP-NUDFAPK ~
$ ./ex1 & ps
[7] 1834
hello
  PID TTY          UID TIME COMMAND
    1828   pts/0    197609 22:37:24 /cygdrive/c/User
s/kyumin/AppData/Roaming/SPB_Data/ex1
    1814   pts/0    197609 22:16:50 /cygdrive/c/User
s/kyumin/AppData/Roaming/SPB_Data/ex1
    1784   pts/0    197609 19:58:59 /usr/bin/bash
    1834   pts/0    197609 22:38:27 /cygdrive/c/User
s/kyumin/AppData/Roaming/SPB_Data/ex1
    1832   pts/0    197609 22:37:52 /cygdrive/c/User
s/kyumin/AppData/Roaming/SPB_Data/ex1
    1830   pts/0    197609 22:37:32 /cygdrive/c/User
s/kyumin/AppData/Roaming/SPB_Data/ex1
    1812   pts/0    197609 22:16:35 /cygdrive/c/User
s/kyumin/AppData/Roaming/SPB_Data/ex1
    1783   ?        197609 19:58:59 /usr/bin/mintty
    1835   pts/0    197609 22:38:27 /usr/bin/ps
```

&이란 명령어를 앞의 명령어를 백그라운드로 동작시킬 때 사용한다. 위 사진의 경우 hello 메시지가 출력되고, 현재 프로세스를 볼 수 있다. ex1 들이 실행되고 있음을 볼 수 있다.

16) Kill it with "kill" command.

```
kyumin@DESKTOP-NUDFAPK ~
$ kill 1830 1812

kyumin@DESKTOP-NUDFAPK ~
$ ps
  PID   PPID   PGID   WINPID   TTY        UID    STIME COMMAND
  1865   1784   1865   12268   pty0       197609 23:00:26 /usr/bin/ps
  1784   1783   1784   18856   pty0       197609 19:58:59 /usr/bin/bash
  1783    1     1783   17908   ?          197609 19:58:59 /usr/bin/mintty
[1]-  Terminated
[4]+  Terminated
      ./ex1
```

Kill 명령어를 사용하여 ex1 실행 중이던 프로세스를 제거했고, ps 를 통해 ex1 이 조회되지 않는 것을 볼 수 있다.

17) Run the program again without & at the end. Open another login window, find out the process ID of the process running in the first window, and kill it.

```
kyumin@DESKTOP-NUDFAPK ~
$ ./ex1
hello

kyumin@DESKTOP-NUDFAPK ~
$ ps
  PID   PPID   PGID   WINPID   TTY        UID    STIME COMMAND
  1784   1783   1784   18856   pty0       197609 19:58:59 /usr/bin/bash
  1872   1868   1872   9932    pty1       197609 23:02:35 /usr/bin/ps
  1868   1867   1868   17516   pty1       197609 23:02:20 /usr/bin/bash
  1867    1     1867    976    ?          197609 23:02:20 /usr/bin/mintty
  1866   1784   1866   15400   pty0       197609 23:02:16 /cygdrive/c/User
s/kyumin/AppData/Roaming/SPB_Data/ex1
  1783    1     1783   17908   ?          197609 19:58:59 /usr/bin/mintty

kyumin@DESKTOP-NUDFAPK ~
$ ./ex1
helloTerminated

kyumin@DESKTOP-NUDFAPK ~
$

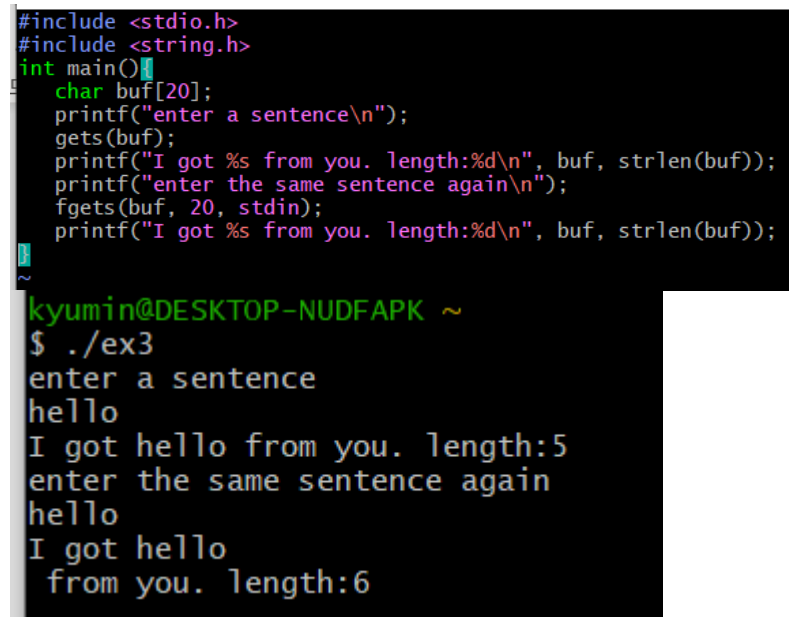
kyumin@DESKTOP-NUDFAPK ~
$ kill 1874

kyumin@DESKTOP-NUDFAPK ~
$ ps
  PID   PPID   PGID   WINPID   TTY        UID    STIME COMMAND
  1784   1783   1784   18856   pty0       197609 19:58:59 /usr/bin/bash
  1868   1867   1868   17516   pty1       197609 23:02:20 /usr/bin/bash
  1867    1     1867    976    ?          197609 23:02:20 /usr/bin/mintty
  1876   1868   1876    9664   pty1       197609 23:05:36 /usr/bin/ps
  1783    1     1783   17908   ?          197609 19:58:59 /usr/bin/mintty
```

Ex1 프로그램을 실행 중인 상태로 다른 터미널 창을 열어서 ps 명령어로 프로세스 확인을 해봤다. Ex1 이 실행 중임을 볼 수 있다. Kill 명령어로 ex1 을 종료했다. 기존 터미널 창에서는 무한루프가 종료된 것을 볼 수 있고, 두 번째 터미널 창에서 ps 로 ex1 이 조회되지 않는 것을 볼 수 있다.

18) Run following and tell the difference between gets and fgets

```
#include <stdio.h>
#include <string.h>
int main(){
    char buf[20];
    printf("enter a sentenceWn");
    gets(buf);
    printf("I got %s from you. length:%dWn", buf, strlen(buf));
    printf("enter the same sentence againWn");
    fgets(buf, 20, stdin);
    printf("I got %s from you. length:%dWn", buf, strlen(buf));
}
```



```
#include <stdio.h>
#include <string.h>
int main()
{
    char buf[20];
    printf("enter a sentence\n");
    gets(buf);
    printf("I got %s from you. length:%d\n", buf, strlen(buf));
    printf("enter the same sentence again\n");
    fgets(buf, 20, stdin);
    printf("I got %s from you. length:%d\n", buf, strlen(buf));
}

kyumin@DESKTOP-NUDFAPK ~
$ ./ex3
enter a sentence
hello
I got hello from you. length:5
enter the same sentence again
hello
I got hello
from you. length:6
```

문자를 입력하고 엔터를 누른다면 버퍼에는 줄바꿈(Wn)으로 들어간다.

Gets 의 경우 줄바꿈(Wn)을 W0(종료)로 바꿔준다. 그렇기 때문에 printf 사용 시 줄바꿈 없이 출력이 된다. 반면 fgets 의 경우 Wn 뒤에 W0(종료)가 들어간다. 그래서 printf 를 해보면 줄바꿈이 생기는 것을 볼 수 있다.

위 사진의 경우 “hello”라는 문자를 입력 후 엔터를 눌렀다. Gets 의 경우 “hello”만을 입력받았고, 문자 길이는 7 로 확인이 된다. fgets 의 경우 “helloWn”을 입력받았기 때문에 줄바꿈이 되었고, 문자 길이도 8 이다.

또한 20글자 이상 입력 시 gets는 모든 글자를 입력 받지만 fgets는 20글자 미만만 입력받는다.

19) Write a program to read a sentence and echo it as follows. Use gets() or fgets(). Search Internet to find out the usage of them (or Do "man gets" or "man fgets").

Enter a sentence

aa bcd e e ff aa bcd bcd hijk lmn al bcd

You entered aa bcd e e ff aa bcd bcd hijk lmn al bcd

```
#include <stdio.h>
#include <string.h>
int main() {
    char buf[41];
    printf("enter a sentence\n");
    fgets(buf, sizeof buf, stdin);
    if (buf[strlen(buf) - 1] == '\n') buf[strlen(buf) - 1] = '\0';

    puts(buf);

    return 0;
}
~
~
~
kyumin@DESKTOP-NUDFAPK ~
$ vi ex3.c

kyumin@DESKTOP-NUDFAPK ~
$ g++ -o ex3 ex3.c

kyumin@DESKTOP-NUDFAPK ~
$ ./ex3
enter a sentence
aa bcd e e ff aa bcd bcd hijk lmn al bcd
aa bcd e e ff aa bcd bcd hijk lmn al bcd
```

Gets 함수는 오버플로우가 발생할 수 있다는 단점이 있다. 그래서 fgets 를 사용하여 코드를 작성했다.

우선 40 글자까지 입력 받고 싶으나 fgets 의 경우 버퍼에 Wn(엔터)까지 입력이 되니 char buf[41]로 지정해줬다. Fgets 로 입력을 받도록 했고, if 문을 사용하여 버퍼 끝에 있는 Wn(엔터)를 W0(종료)로 바꿔주었다. 이후 buf 를 출력할 수 있도록 puts 를 사용했다. 결과를 확인해보면 줄바꿈 없이 정상적으로 출력되는 모습을 볼 수 있다.

20) Show the first 20 bytes of ex1.c in Problem 4 with xxd. Interpret them.

```
kyumin@DESKTOP-NUDFAPK ~
$ xxd ex1.c
00000000: 2369 6e63 6c75 6465 203c 7374 6469 6f2e  #include <stdio.
00000010: 683e 0a76 6f69 6420 6d61 696e 2829 7b0a  h>.void main(){.
00000020: 2020 2070 7269 6e74 6628 2268 656c 6c6f   printf("hello
00000030: 5c6e 2229 3b0a 7d0a                        \n");.}.
```

제어 문자		공백 문자		구두점		숫자		알파벳			
10진	16진	문자	10진	16진	문자	10진	16진	문자	10진	16진	문자
0	0x00	NUL	32	0x20	SP	64	0x40	@	96	0x60	
1	0x01	SOH	33	0x21	!	65	0x41	A	97	0x61	a
2	0x02	STX	34	0x22	"	66	0x42	B	98	0x62	b
3	0x03	ETX	35	0x23	#	67	0x43	C	99	0x63	c
4	0x04	EOT	36	0x24	\$	68	0x44	D	100	0x64	d
5	0x05	ENQ	37	0x25	%	69	0x45	E	101	0x65	e
6	0x06	ACK	38	0x26	&	70	0x46	F	102	0x66	f
7	0x07	BEL	39	0x27	'	71	0x47	G	103	0x67	g
8	0x08	BS	40	0x28	(	72	0x48	H	104	0x68	h
9	0x09	HT	41	0x29	)	73	0x49	I	105	0x69	i
10	0x0A	LF	42	0x2A	*	74	0x4A	J	106	0x6A	j
11	0x0B	VT	43	0x2B	+	75	0x4B	K	107	0x6B	k
12	0x0C	FF	44	0x2C	,	76	0x4C	L	108	0x6C	l
13	0x0D	CR	45	0x2D	-	77	0x4D	M	109	0x6D	m
14	0x0E	SO	46	0x2E	.	78	0x4E	N	110	0x6E	n
15	0x0F	SI	47	0x2F	/	79	0x4F	O	111	0x6F	o
16	0x10	DLE	48	0x30	0	80	0x50	P	112	0x70	p
17	0x11	DC1	49	0x31	1	81	0x51	Q	113	0x71	q
18	0x12	DC2	50	0x32	2	82	0x52	R	114	0x72	r
19	0x13	DC3	51	0x33	3	83	0x53	S	115	0x73	s
20	0x14	DC4	52	0x34	4	84	0x54	T	116	0x74	t
21	0x15	NAK	53	0x35	5	85	0x55	U	117	0x75	u
22	0x16	SYN	54	0x36	6	86	0x56	V	118	0x76	v
23	0x17	ETB	55	0x37	7	87	0x57	W	119	0x77	w
24	0x18	CAN	56	0x38	8	88	0x58	X	120	0x78	x
25	0x19	EM	57	0x39	9	89	0x59	Y	121	0x79	y
26	0x1A	SUB	58	0x3A	:	90	0x5A	Z	122	0x7A	z
27	0x1B	ESC	59	0x3B	;	91	0x5B	[	123	0x7B	{
28	0x1C	FS	60	0x3C	<	92	0x5C	\	124	0x7C	
29	0x1D	GS	61	0x3D	=	93	0x5D	]	125	0x7D	}
30	0x1E	RS	62	0x3E	>	94	0x5E	^	126	0x7E	~
31	0x1F	US	63	0x3F	?	95	0x5F	_	127	0x7F	DEL

문제 4 번의 코드 내용이 들어간 ex1.c 를 xxd 명령어를 사용하여 확인했다. 첫 20 바이트는 00000000: 2369 6e63 6c75 6465 203c 이다. 여기서 한글자는 2 바이트씩이다.

2369 = #i / 6e63 = nc / 6c75 = lu / 6465 = de / 203c = 공백< 이다.

Xxd 를 통해 16 진수로 내용을 확인할 수 있다.

21) (For true Linux) [ELF format] An executable files in Linux follows ELF (Executable and Linkable Format) format as below. Show the first 20 bytes of ex1, the executable file (not ex1.c) in Problem 4, with xxd. Interpret them.

ELF format= ELF header + Program header table + Section 1 + Section 2 + ... + Section n + Section header table

ELF header =

e\_ident(16)+ e\_type(2)+ e\_machine(2)+ e\_version(4)+ e\_entry(4)+ e\_phoff(4)+ e\_shoff(4)+  
e\_flags(4)+ e\_ehsize(2)+ e\_phentsize(2)+ e\_phnum(2)+ e\_shentsize(2)+ e\_shnum(2)+  
e\_shstrndx(2)

e\_ident=7f E L F + EI\_CLASS(1) + EI\_DATA(1) + EI\_VERSION(1) + EI\_OSABI(1) + EI\_ABIVERSION(1) + EI\_PAD(7)

EI\_CLASS = 1 if 32bit application or 2 if 64bit application

EI\_DATA = 1 if little endian or 2 if big endian

EI\_VERSION = 1

EL\_OSABI = 0 for System V, 1 for HP-UX, 2 for NetBSD, 3 for Linux, 4 for GNU Hurd, ...  
EL\_ABIVERSION = depends on ABI version  
EI\_PAD = 9 zero's

e\_type= 1 for relocatable file, 2 for executable file, 3 for shared object file  
e\_machine = 3 for x386, 0x32 for IA-64, 0x3e for amd64, ...  
e\_version = 1  
e\_entry = program starting address

.....

(\* Refer to [https://en.wikipedia.org/wiki/Executable\\_and\\_Linkable\\_Format](https://en.wikipedia.org/wiki/Executable_and_Linkable_Format) for the rest of ELF file format)

Cygwin 을 사용함.

22) (For cygwin) [PE format] Executable file in PC is in PE format. Write ex1.c in cygwin and compile with gcc. Show the first 20 bytes of ex1.exe, the executable file (not ex1.c) with xxd. Interpret them. An executable file in Windows follows PE (Portable Executable) format as below (for detail of PE format, refer to Internet):

PE=dos header + image nt header + section table + sections

dos header = image dos header + dos stub

image dos header(64 byte)=

0,1: e\_magic(2) : 4d 5a

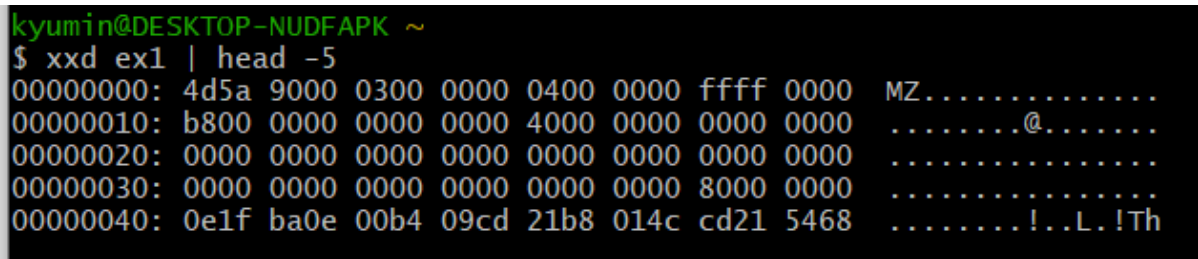
2,3: e\_cblp(2)

.....

3c-3f:e\_lfanew(4) : offset of IMAGE\_NT\_HEADER

image nt header=pe signature(4) + file header + optional header

.....



```
kyumin@DESKTOP-NUDFAPK ~  
$ xxd ex1 | head -5  
00000000: 4d5a 9000 0300 0000 0400 0000 ffff 0000  MZ.....  
00000010: b800 0000 0000 0000 4000 0000 0000 0000  .....@.....  
00000020: 0000 0000 0000 0000 0000 0000 0000 0000  .....  
00000030: 0000 0000 0000 0000 0000 0000 8000 0000  .....  
00000040: 0e1f ba0e 00b4 09cd 21b8 014c cd21 5468  .....!..L.!Th
```

Ex1 은 기계어로 변환된 내용이 들어갔기 때문에 내용이 매우 길다. 그래서 | head -5 를 뒤에 붙여서 5 줄만 출력되도록 했다. 그 중 20 바이트인 부분은 00000000: 4d5a 9000 0300 0000 0400 까지다. 16 진수를 아스키 코드와 비교해보면 4d5a = MZ 이지만 뒷부분은 90 처럼 아스키코드에 없거나 00, 03 처럼 제어문자라서 위 사진의 우측 부분에는 . 으로 표기된다. Ex1 파일의 내용은 기계어이기 때문에 xxd 를 통해 16 진수로 내용 확인이 가능할 뿐이다.

23) (For Mac) Do 21) with Mach-O format.

Cygwin을 사용함