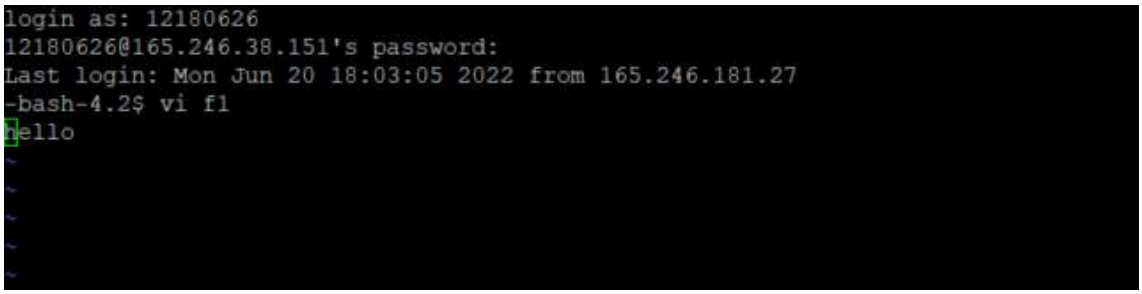
lect3-4 HW

시스템프로그래밍 1 분반 12180626 성시열

lect 3

2. Homework

1) Make a file with vi.



vi f1 을 입력하여 f1 파일을 vi 편집기를 통해 열어준다. open 직후의 상태는 command 모드이고, f1 에 저장되어있던 hello 가 문자열로 출력된다.

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

command 모드에서 'i'버튼을 통해 INSERT 모드로 변환하였다. 아래 INSERT 라는 문구를 보고 insert 모드임을 확인하였다. 이 모드에서는 문자열을 입력할 수 있다.

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

This is the second line of text.

라는 문자를 직접 입력하였다.

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

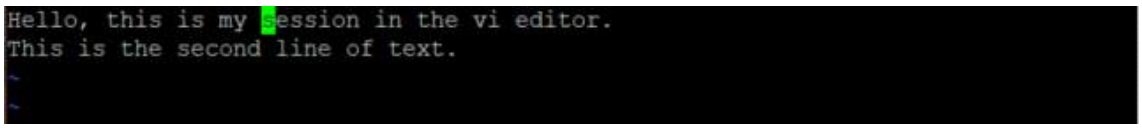
입력을 마친 뒤 esc 버튼을 누르고 :wq 를 입력한다. wq 는 write quick 의 약자로 저장후 빠져나옴을 뜻한다.

enter 를 누르면 입력한 문자열을 저장한 뒤 vi 편집기를 빠져 나왔음을 확인하였다.

4) Reopen the file

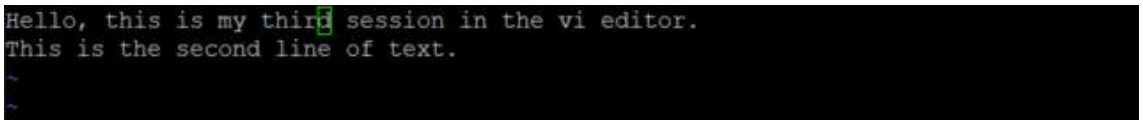
다시 vi 편집기로 파일을 열기 위해 vi f1 명령어를 입력하였다. 실행하면 위에서 입력했던 문자열이 저장되어 출력됨을 확인할 수 있다.

5) Use 'j', 'k', 'h', 'l' to move the cursor around. Move the cursor to the word "first". Use 'x' key to delete the word "first". The result should be as follows.



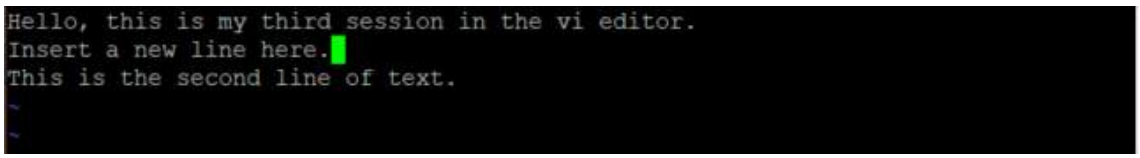
command 모드에서 화살표가 아닌 'j'(down), 'k'(up), 'h'(left), 'l'(right)를 사용하여 커서를 움직여보고 x 키를 사용하여 'first '라는 문자열을 삭제한다.

6) Insert "third" as follows.



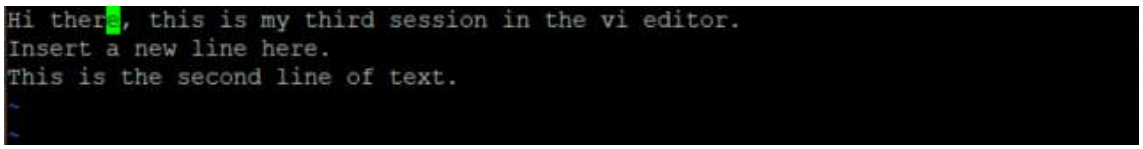
command 모드에서 'i' 버튼을 통해 insert 모드로 변경하고 해당 위치에 'third'라는 문자열을 입력하고 esc 를 통해 빠져나온다.

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



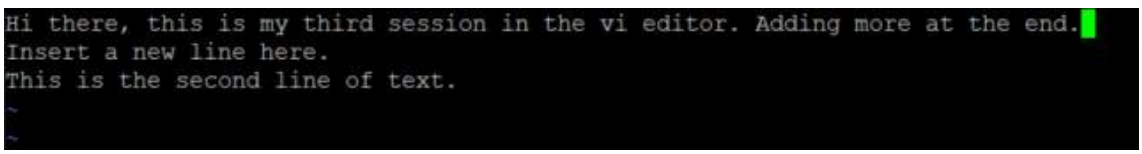
command 모드에서 'o' 버튼을 누르게 되면 줄바꿈과 동시에 insert 모드로 변경된다. 이후 'Insert a new line here.' 이라는 문자열을 입력한 뒤 esc 를 통해 빠져나온다.

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



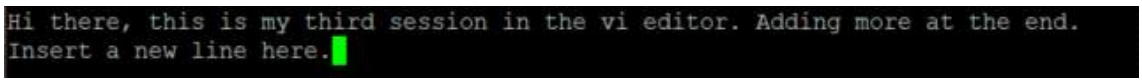
command 모드에서 'x' 버튼을 통해 'Hello'를 지우고, 'i'키를 통해 insert 모드로 변경하여 'Hi there'을 입력한다. 이후 esc 버튼을 통해 빠져나온다.

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



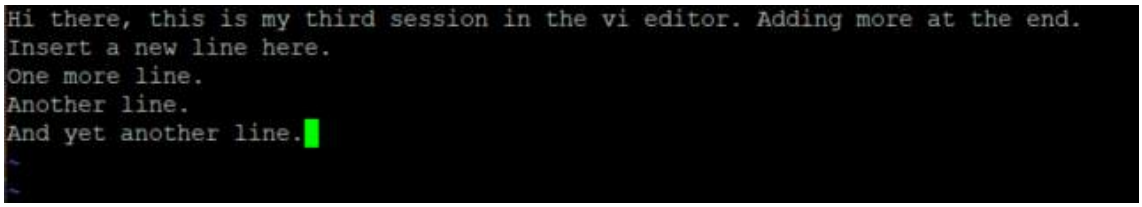
command 모드에서 'a' 버튼을 누르게 되면 현재 커서에서부터 insert 모드를 시작한다. 이후 'Adding more at the end.' 라는 문자열을 입력한 뒤 esc 버튼을 통해 빠져나온다.

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



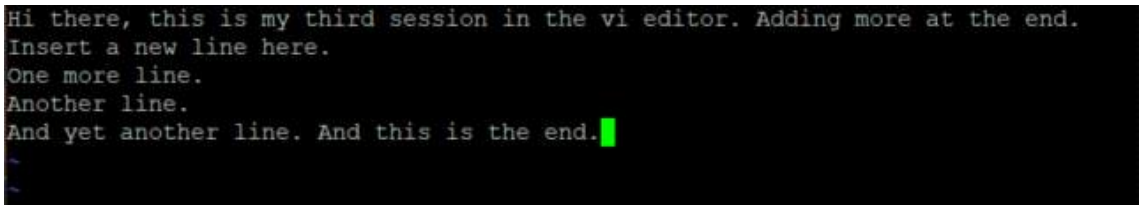
command 모드에서 삭제하고 싶은 line 에 커서를 옮긴 뒤 'dd'(delete a line)를 누르게 되면 해당 line 이 삭제된다. 'dd' 버튼을 통해 마지막 line 을 삭제해주었다.

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



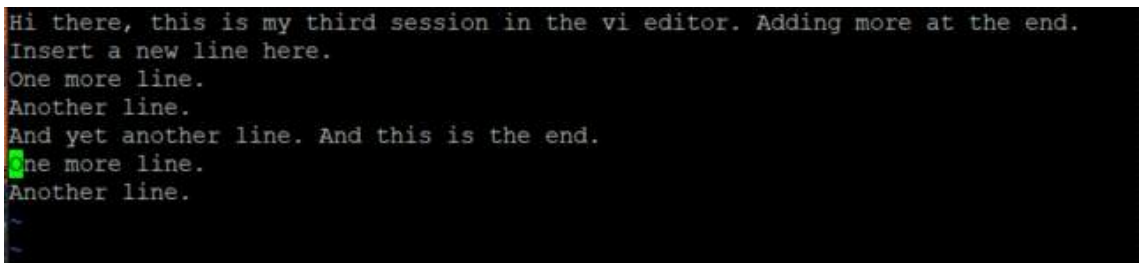
command 모드에서 'o' 버튼을 누르게 되면 줄바꿈과 동시에 insert 모드로 변경된다. 이후 'One more line.
Another line.
And yet another line.' 이라는 문자열을 입력한 뒤 esc 를 통해 빠져나온다.

12) Change the last line.



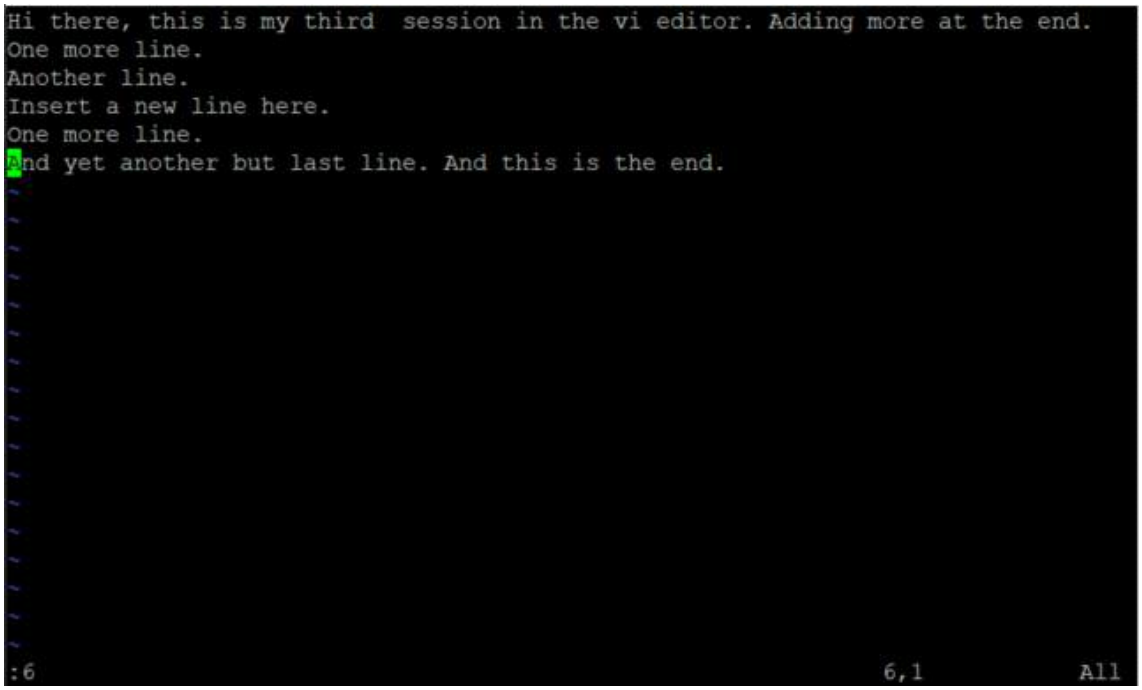
command 모드에서 'i'키를 통해 insert 모드로 변경하여 마지막 줄에 ' And this is the end.'를 추가한다.

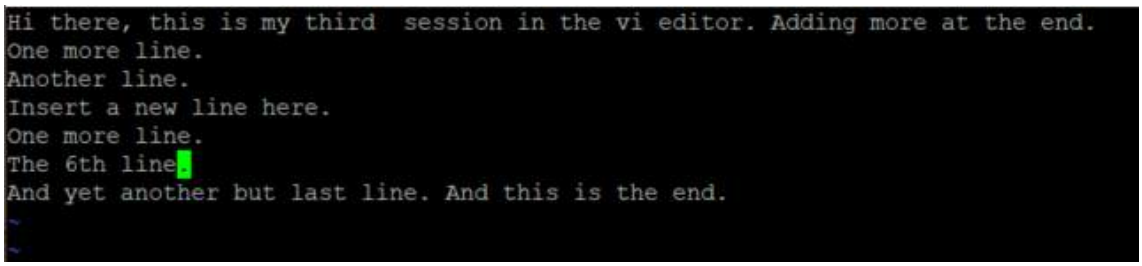
13) Copy and paste as follows. Use '2yy' to copy two lines at the current cursor; move the cursor to another location and use 'p' to paste them at that location.



command 모드에서 '2yy' 키를 통해 현재 커서로부터 2 줄을 복사할 수 있고, 이후 'p' 키를 눌러 붙여넣기를 할 수 있다. 'One more line.' 문자열에 커서를 두고 '2yy' 명령어를 입력한 뒤 문장의 끝으로 가서 p 를 누르면 위와 같이 2 줄이 복사됨을 확인할 수 있다.

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





command 모드에서 :6 을 입력한 뒤 enter 누르면 커서가 6 번째 줄의 처음으로 이동한다. 'o' 버튼을 누르게 되면 줄바꿈과 동시에 insert 모드로 변경된다. 이후 'The 6th line.'를 추가한다.

위에서 입력한 문자들을 저장하기 위해 command 상태에서 :wq 를 실행하여 저장후 빠져나온다.

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

vi ex1.c 를 통해 exl.c 라는 새로운 파일을 열어준다.

'i' 버튼을 통해 insert 모드로 변경한 뒤 간단한 c 프로그래밍을 문자열로 입력하였다. 이후 esc 버튼과 :wq 를 실행시켜 저장한 뒤 빠져나온다.

cat 명령어를 통해 ex1.c 에 저장된 문자열을 그대로 출력해줌으로써 작성한 c 프로그래밍이 잘 저장이 되었는지 확인하였다.

gcc 는 c 프로그래밍을 컴파일 하는데 필요한 명령어이다. gcc -o ex1 ex1.c 를 통해 ex1.c 파일에 저장된 문자열을 기계어로 변환하는 컴파일 과정을 거치고 변환된 기계어를 ex1 에 저장한다.

./ex1 을 통해 ex1 에 저장된 기계어를 Run 하면 hi there 이 print 됨을 확인할 수 있다.

lect 4

2. Homework

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

```
-bash-4.2$ cd
-bash-4.2$ pwd
/home/sp5/12180626
-bash-4.2$ ls
d1  d2  ex1  ex1.c  f1  f3  f4  f5  f6  f7
-bash-4.2$ ls -l
total 44
d--------- 2 12180626 12180626 4096 Jun 20 10:50 d1
drwxrwxr-x 2 12180626 12180626 4096 Jun 20 15:03 d2
-rwxrwxr-x 1 12180626 12180626 8501 Jun 21 02:31 ex1
-rw-rw-r-- 1 12180626 12180626   56 Jun 21 02:27 ex1.c
-rw-rw-r-- 1 12180626 12180626  212 Jun 21 02:24 f1
-rw-rw-r-- 1 12180626 12180626    6 Jun 20 11:18 f3
-rw-rw-r-- 1 12180626 12180626    6 Jun 20 11:17 f4
-rw-rw-r-- 2 12180626 12180626    6 Jun 20 18:10 f5
-rw-rw-r-- 2 12180626 12180626    6 Jun 20 18:10 f6
-rw-rw-r-- 1 12180626 12180626    0 Jun 20 18:32 f7
```

cd 를 통해 현재 디렉토리를 로그인 디렉토리로 이동시킨다.

pwd 를 통해 현재 위치의 디렉토리를 확인한다. 12180626 디렉토리에 위치함을 확인하였다.

ls 와 ls –l 을 통해 12180626 디렉토리의 하위 모든 파일들을 확인한다. d1, d2 두 개의 디렉토리와 ex1, ex1.c, f1, f3, f4, f5, f6, f7 8 개의 파일이 있음을 확인하였다.

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.

```
-bash-4.2$ cd /etc
-bash-4.2$ file *
abrt:                   directory
adjtime:                ASCII text
akonadi:                directory
aliases:                ASCII text
aliases.db:             regular file, no read permission
alsa:                   directory
alternatives:           directory
anacrontab:             ASCII text
ant.conf:               ASCII text
ant.d:                  directory
anthy-conf:             ASCII text
asound.conf:            ASCII text
at.deny:                very short file (no magic)
atmsigd.conf:           ASCII text
at-spi2:                directory
audisp:                 directory
audit:                  directory
avahi:                  directory
bash_completion.d:      directory
bashrc:                 ASCII text
binfmt.d:               directory
bluetooth:              directory
bonobo-activation:      directory
brltty:                 directory
brltty.conf:            assembler source, UTF-8 Unicode text
capi20.conf:            ASCII text
```

(출력 결과가 너무 길어 일부 생략하여 첨부하였습니다.)

절대경로를 통해 etc 라는 디렉토리로 이동하였다. 이후 file *을 통해 모든 파일의 정보를 확인하였다.

```
-bash-4.2$ file * | grep text
adjtime:                ASCII text
aliases:                ASCII text
anacrontab:             ASCII text
ant.conf:               ASCII text
anthy-conf:             ASCII text
asound.conf:            ASCII text
atmsigd.conf:           ASCII text
bashrc:                 ASCII text
brltty.conf:            assembler source, UTF-8 Unicode text
capi20.conf:            ASCII text
capi.conf:              ASCII text
chrony.conf:            ASCII text
colord.conf:            ASCII text
crontab:                ASCII text
csh.cshrc:              ASCII text
csh.login:              ASCII text
DIR_COLORS:             ASCII text
DIR_COLORS.256color:    ASCII text
DIR_COLORS.lightbgcolor: ASCII text
dnsmasq.conf:           ASCII text
dracut.conf:            ASCII text
drirc:                  ASCII text
e2fsck.conf:            ASCII text
ethertypes:             ASCII text
fedora-release:         ASCII text
filesystems:            ASCII text
fprintd.conf:           ASCII text
```

(출력 결과가 너무 길어 일부 생략하여 첨부하였습니다.)

file * | grep text 를 활용하여 모든 파일 중 text 라는 특정 문자열이 포함된 file 들의 정보만을 출력한다.

3) 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" commad.

```
-bash-4.2$ whereis passwd
passwd: /bin/passwd /usr/bin/passwd /etc/passwd /usr/share/man/man5/passwd.5.gz
/usr/share/man/man1/passwd.1.gz
-bash-4.2$ whereis stdio.h
stdio: /usr/include/stdio.h /usr/share/man/man3/stdio.3.gz
-bash-4.2$ whereis ls
ls: /bin/ls /usr/bin/ls /usr/share/man/man1p/ls.1p.gz /usr/share/man/man1/ls.1.g
z
```

whereis 명령어는 파일 이름을 통해 파일 위치(경로)를 찾아주는 명령어이다. 'passwd', 'stdio.h', 'ls'의 위치를 찾아 출력하였다.

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

```
-bash-4.2$ cd
-bash-4.2$ vi ex1.c
```

cd 를 통해 로그인 디렉토리로 이동한 뒤, vi ex1.c 를 입력한다.

v1 ex1.c 를 실행 시킴과 동시에 ex1.c 의 command mode 로 접속된다. 이후 'i'키를 눌러 insert 모드로 변경한 뒤

#include <stdio.h>

void main(){

printf("hello₩n");

}

를 입력한다. esc 버튼을 통해 insert 모드를 빠져나오고 :wq 를 통해 문자열을 exl.c 에 저장한 뒤 파일을 빠져나온다.

이후 c 프로그래밍으로 작성된 ex1.c 파일을 컴파일하고 변환된 기계어를 ex1 에 저장하기 위해 gcc -o ex1 ex1.c 를 실행시킨다.

이후 코드가 잘 컴파일되었는지 확인하기 위해 ./ex1 를 통해 Run 시킨다. 위에서 작성해준 C 프로그래밍의 결과값인 'hello'가 잘 출력됨을 확인하였다.

g++을 활용한 컴파일을 진행해보기 위해 c++ 프로그래밍으로 코드를 작성해본다.

다시 vi ex1.c 를 통해 ex1.c 파일을 vi 편집기를 활용해 open 한다.

ex1.c 파일에

#include <stdio.h>

int main(){

printf("hello₩n);

}

를 입력한다. (void main → int main)

이후 esc 버튼을 통해 command 모드로 변경한 뒤 :wq 를 통해 문자열을 저장하고 파일을 빠져나온다.

이후 c++ 프로그래밍으로 작성된 ex1.c 파일을 컴파일하고 변환된 기계어를 ex1 에 저장하기 위해 g++ -o ex1 ex1.c 를 실행시킨다.

이후 코드가 잘 컴파일되었는지 확인하기 위해 ./ex1 를 통해 Run 시킨다. 위에서 작성해준 C++ 프로그래밍의 결과값인 'hello'가 잘 출력됨을 확인하였다.

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>".

```
-bash-4.2$ cat ex1.c
#include <stdio.h>
int main(){
printf("hello\n");
}
```

cat ex1.c 를 통해 ex1.c 에 저장된 문자열을 그대로 출력해주었고, 입력했던 문자열과 일치하는 것을 확인하였다.

```
-bash-4.2$ xxd ex1.c
0000000: 2369 6e63 6c75 6465 203c 7374 6469 6f2e  #include <stdio.
0000010: 683e 0a69 6e74 206d 6169 6e28 297b 0a70  h>.int main(){.p
0000020: 7269 6e74 6628 2268 656c 6c6f 5c6e 2229  rintf("hello\n")
0000030: 3b0a 7d0a                                ;.}.
```

xxd ex1.c 를 통해 ex1.c 에 저장된 문자열을 16 진수로 출력해주었다.
'#include<stdio.h>'의 ASCII 코드는 2369 6e63 6465 203c 7374 6469 6f2e 683e 임을 확인하였다.
xxd ex1.c 의 결과값을 왼쪽 콜론 전과 가운데 16 진수, 오른쪽 문자열 총 3 부분으로 나누었을 때, 왼쪽과 오른쪽 부분은 사용자 편의를 위해 제공해주는 정보이며 가운데 부분이 실제 가지고 있는 16 진수임을 알 수 있다.

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

```
-bash-4.2$ cat ex1
`  /lib64/ld-linux-x86-64.so.2GNU GNU\;) I▒
#;▒▒o>
▒'▒▒  ▒▒3 O libstdc++.so.6__gmon_start___Jv_RegisterClasses_ITM_deregisterTMClon
eTable_ITM_registerTMCloneTablelibm.so.6libgcc_s.so.1libc.so.6puts__libc_start_m
ainGLIBC_2.2.5▒ui       ▒▒▒`` `(`H▒H▒M
                                  H▒▒t▒▒▒▒5B
                                          ▒▒%D
                                              @▒▒%B
                                                   h▒▒▒▒▒▒▒▒%:
                                                            h▒▒▒▒▒▒▒%2
                                                                    h▒
▒▒▒▒▒1▒I▒▒^H▒▒H▒▒▒PTI▒▒@H▒▒@H▒▒▒@▒▒▒▒▒▒f▒@▒?`UH-8`H▒▒H▒▒w]øH▒▒t▒]▒8`▒▒▒▒▒8`UH-8`H▒
▒H▒▒H▒▒H▒▒?H▒H▒▒H▒▒u]úH▒▒t▒]▒8`▒▒▒▒▒=▒
 uUH▒▒▒▒~▒▒▒▒]▒z
`H▒▒▒▒▒]▒{▒▒▒▒v▒▒▒f▒UH▒▒▒@▒▒▒▒▒▒▒]▒f.▒DH▒l$▒L▒d$▒H▒-▒ L▒%▒ H▒\$▒L▒l$▒L▒t$▒L▒|$▒H▒▒▒
8L)▒A▒▒I▒▒H▒▒I▒▒1▒▒I▒▒▒H▒▒t@L▒▒L▒▒D▒▒A▒▒H▒▒H9▒u▒H▒\H▒l$L▒d$L▒l$ L▒t$(L▒|$0H▒▒▒8▒▒▒
▒▒f▒H▒H▒▒hello4▒▒▒▒▒8▒▒▒P$▒▒▒▒H▒▒▒▒▒▒▒▒▒zRx
▒▒▒▒▒*zRx
$p▒▒▒▒@FJ
P       ▒?;*3$"Dt▒▒▒A▒C
$dx▒▒▒▒▒J▒▒f@▒▒▒▒▒▒▒▒▒@▒@is▒
```

(출력 결과가 너무 길어 일부 생략하여 첨부하였습니다.)

cat 명령어는 저장된 ASCII 값을 문자열로 출력하는 명령어이다. cat ex1 은 ex1 에 저장된 값을 문자열로 출력해줘야하는데, ex1 에 저장된 값은 ex1.c 파일을 컴파일하고 변환된 기계어가 저장된 파일이므로 위와 같이 출력값이 깨지는 것을 확인하였다.

6-1) The compiler has translated the C statements in ex1.c into machine instructions and stored in ex1:

```
-bash-4.2$ xxd ex1 > x
-bash-4.2$ vi x
```

xxd 명령어는 저장된 값을 16 진수로 출력해주는 함수이다. xxd ex1 > x 를 통해 ex1 에 저장된 값을 16 진수로 바꿔주고 그 값을 x 에 저장하였다. 이후 vi 편집기를 통해 x 파일을 열어주었다.

```
0000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
0000010: 0200 3e00 0100 0000 0005 4000 0000 0000  ..>.......@.....
0000020: 4000 0000 0000 0000 6811 0000 0000 0000  @.......h.......
0000030: 0000 0000 4000 3800 0900 4000 1e00 1b00  ....@.8...@.....
0000040: 0600 0000 0500 0000 4000 0000 0000 0000  ........@.......
0000050: 4000 4000 0000 0000 4000 4000 0000 0000  @.@.....@.@.....
0000060: f801 0000 0000 0000 f801 0000 0000 0000  ................
0000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
0000080: 3802 0000 0000 0000 3802 4000 0000 0000  8.......8.@.....
0000090: 3802 4000 0000 0000 1c00 0000 0000 0000  8.@.............
00000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
00000b0: 0100 0000 0500 0000 0000 0000 0000 0000  ................
00000c0: 0000 4000 0000 0000 0000 4000 0000 0000  ..@.......@.....
00000d0: d407 0000 0000 0000 d407 0000 0000 0000  ................
00000e0: 0000 2000 0000 0000 0100 0000 0600 0000  .. ............
00000f0: e00d 0000 0000 0000 e00d 6000 0000 0000  ..........`.....
0000100: e00d 6000 0000 0000 5402 0000 0000 0000  ..`.....T.......
0000110: 5802 0000 0000 0000 0000 2000 0000 0000  X......... .....
0000120: 0200 0000 0600 0000 f80d 0000 0000 0000  ................
0000130: f80d 6000 0000 0000 f80d 6000 0000 0000  ..`.......`.....
0000140: 0002 0000 0000 0000 0002 0000 0000 0000  ................
0000150: 0800 0000 0000 0000 0400 0000 0400 0000  ................
0000160: 5402 0000 0000 0000 5402 4000 0000 0000  T.......T.@.....
0000170: 5402 4000 0000 0000 4400 0000 0000 0000  T.@.....D.......
0000180: 4400 0000 0000 0000 0400 0000 0000 0000  D...............
0000190: 50e5 7464 0400 0000 c806 0000 0000 0000  P.td............
00001a0: c806 4000 0000 0000 c806 4000 0000 0000  ..@.......@.....
00001b0: 3400 0000 0000 0000 3400 0000 0000 0000  4.......4.......
00001c0: 0400 0000 0000 0000 51e5 7464 0600 0000  ........Q.td....
00001d0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001e0: 0000 0000 0000 0000 0000 0000 0000 0000  ................
00001f0: 0000 0000 0000 0000 0800 0000 0000 0000  ................
0000200: 52e5 7464 0400 0000 e00d 0000 0000 0000  R.td............
0000210: e00d 6000 0000 0000 e00d 6000 0000 0000  ..`.......`.....
0000220: 2002 0000 0000 0000 2002 0000 0000 0000   ....... .......
                                                  1,1           Top
```

(출력 결과가 너무 길어 일부 생략하여 첨부하였습니다.)

vi 편집기로 open 한 x 파일을 확인하면 위와 같다

```
00005e0: e97b ffff ffe9 76ff ffff 6690 5548 89e5  .{....v...f.UH..
00005f0: bfc0 0640 00e8 e6fe ffff b800 0000 005d  ...@...........]
0000600: c366 2e0f 1f84 0000 0000 000f 1f44 0000  .f...........D..
```

검색 기능을 활용해 '/5548'을 검색해본 결과 '55 48 89 e5 bf c0 06 40 00'을 00005e0 ~ 00005f0 번째 줄에서 찾아볼 수 있었다.

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

```
-bash-4.2$ ls
d1  d2  ex1  ex1.c  f1  f3  f4  f5  f6  f7  x
-bash-4.2$ cp ex1.c ex2.c
```

```
-bash-4.2$ cp ex1.c ex3.c
-bash-4.2$ cp ex1.c ex4.c
-bash-4.2$ rm ex2.c
-bash-4.2$ mv ex3.c y.c
-bash-4.2$ ls
d1  d2  ex1  ex1.c  ex4.c  f1  f3  f4  f5  f6  f7  x  y.c
```

문제에서 요구한대로 cp 를 활용하여 ex1.c 파일을 복사하여 ex2.c, ex3.c, ex4.c 파일로 붙여넣기 하고 rm ex2.c 를 통해 ex2.c 파일을 제거하였다. 이름을 변경하기 위해 mv ex3.c y.c 를 활용하여 ex3.c 파일의 이름을 y.c 로 변경하였다.
ls 명령어를 통해 실행했던 과정들이 잘 포함되어있음을 확인하였다.

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

```
-bash-4.2$ mkdir d3
-bash-4.2$ cp y.c d3
-bash-4.2$ cd d3
-bash-4.2$ ls
y.c
```

mkdir d3 를 통해 d3 이라는 디렉토리를 생성하고
cp y.c d3 를 통해 y.c 파일을 복사하여 d3 디렉토리에 붙여넣기하였다.
이후 d3 디렉토리로 이동한 뒤 ls 를 하여 d3 디렉토리에 있는 파일을 모두 확인하면 복사된 y.c 파일 존재하는 것을 알 수 있다.

9) Redirect the output of ex1 to another file using ">" symbol.

```
-bash-4.2$ cd
-bash-4.2$ ls
d1  d2  d3  ex1  ex1.c  ex4.c  f1  f3  f4  f5  f6  f7  x  y.c
-bash-4.2$ ./ex1 > f1
-bash-4.2$ cat f1
hello
```

cd 명령어를 통해 로그인 디렉토리로 이동하였다. ls 를 통해 로그인 디렉토리에 존재하는 파일들을 확인하였다. '>' 기호는 출력되는 값을 파일에 저장할 수 있는 기호인데, ./ex1 > f1 을 통해 ./ex1 의 출력 결과를 f1 에 저장하고 cat f1 을 통해 f1 에 저장된 값을 문자열로 출력하였다. 이 전에 입력했던 프로그래밍의 출력값과 동일한 'hello'라는 문자열이 출력됨을 확인하였다.

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

```
-bash-4.2$ grep -nr hello *
grep: d1: Permission denied
d2/d2:1:hello
d2/f4:1:hello
d3/y.c:3:printf("hello\n");
Binary file ex1 matches
ex1.c:3:printf("hello\n");
ex4.c:3:printf("hello\n");
f1:1:hello
f3:1:hello
f4:1:hello
x:109:00006c0: 6865 6c6c 6f00 0000 011b 033b 3400 0000  hello......;4...
y.c:3:printf("hello\n");
```

grep 명령어를 통해 hello 가 들어간 문자열을 찾아 보았다. -nr 옵션 중에서 –n 은 문자열이 나온 번호를 알려주는 옵션이고, -r 옵션은 하위 디렉토리를 내려가면서 모두 찾는 옵션이다. * 기호는 디렉토리의 모든 파일에 대해 명령어를 실행한다는 뜻이다.
그 결과 hello 가 포함된 파일들을 모두 찾았고, 몇 번 line 인지도 확인하였다.

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

```
-bash-4.2$ ps -ef
UID        PID  PPID  C STIME TTY          TIME CMD
root         1     0  0 Jun15 ?        00:00:06 /usr/lib/systemd/systemd --switc
root         2     0  0 Jun15 ?        00:00:00 [kthreadd]
root         3     2  0 Jun15 ?        00:00:00 [ksoftirqd/0]
root         5     2  0 Jun15 ?        00:00:00 [kworker/0:0H]
root         7     2  0 Jun15 ?        00:00:00 [kworker/u:0H]
root         8     2  0 Jun15 ?        00:00:00 [migration/0]
root         9     2  0 Jun15 ?        00:00:00 [rcu_bh]
root        10     2  0 Jun15 ?        00:01:05 [rcu_sched]
root        11     2  0 Jun15 ?        00:00:01 [watchdog/0]
root        12     2  0 Jun15 ?        00:00:01 [watchdog/1]
root        13     2  0 Jun15 ?        00:00:00 [migration/1]
root        14     2  0 Jun15 ?        00:00:00 [ksoftirqd/1]
root        16     2  0 Jun15 ?        00:00:00 [kworker/1:0H]
root        17     2  0 Jun15 ?        00:00:01 [watchdog/2]
root        18     2  0 Jun15 ?        00:00:00 [migration/2]
root        19     2  0 Jun15 ?        00:00:00 [ksoftirqd/2]
```

(출력 결과가 너무 길어 일부 생략하여 첨부하였습니다.)
ps-ef 를 사용하여 현재 시스템에 존재하는 모든 유저의 모든 프로세스들을 확인하였다.

12) "ps -ef" shows all the processes in the system. How do you know which ones are yours? 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?

```
-bash-4.2$ tty
/dev/pts/12
```

tty 를 통해 터미널 넘버를 확인한 결과 12 번임을 알 수 있었다.

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

```
-bash-4.2$ vi ex1.c
```

ex1.c 파일을 수정하기 위해 vi exl.c 를 활용하여 vi 편집기로 파일을 open 한다.

```
#include <stdio.h>
void  main(){
int a=0;
int b=0;
printf("Type 2 number\t");
scanf("%d %d", &a, &b);
printf("Sum of them is %d\n", a+b);
}
~
~
```

이후 두 개의 값을 받아 두 값을 더해주는 간단한 C 프로그래밍을 구성한다.

```
-bash-4.2$ gcc -o ex1 ex1.c
-bash-4.2$ ./ex1
Type 2 number   3
7
Sum of them is 10
```

gcc -o ex1 ex1.c 를 통해 ex1.c 파일에 저장된 문자열을 기계어로 변환하는 컴파일 과정을 거치고 변환된 기계어를 ex1 에 저장한다. ./ex1 을 통해 실행을 시켜보면 scanf()로 인해 값을 입력하도록 실행이 되고, 두 값을 입력하면 그 값을 더해준 값을 출력해준다.

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

```
#include <stdio.h>
void  main(){
printf("hello");
fflush(stdout);
for(;;);
}
~
~
```

ex1.c 를 수정하여 hello 가 무한으로 출력되는 C 프로그래밍을 작성하였다. 이후 위와 같은 과정으로 gcc -o ex1 ex1.c 를 통해 ex1.c 파일에 저장된 문자열을 기계어로 변환하는 컴파일 과정을 거치고 변환된 기계어를 ex1 에 저장한다.

```
-bash-4.2$ gcc -o ex1 ex1.c
-bash-4.2$ ./ex1
hello
```

./ex1 을 통해 실행을 시켜보면 fflush 를 통해 출력 버퍼를 바로 비워버려서 hello 를 출력시킨 후 무한 루프에 빠지게 된다.

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.

현재는 무한루프가 빠진 상태이기 때문에 새로운 putty 창을 열어 ps 명령어를 실행해본다.

```
login as: 12180626
12180626@165.246.38.151's password:
Last login: Tue Jun 21 09:06:44 2022 from 165.246.181.57
-bash-4.2$ ./ex1 &
[1] 3496
-bash-4.2$ hello
-bash-4.2$ ps
  PID TTY          TIME CMD
 3356 pts/6    00:00:00 bash
 3496 pts/6    00:00:05 ex1
 3497 pts/6    00:00:00 ps
```

./ex1 &를 사용하면 ./ex1 으로 인해 파일이 실행되고 있음에도 백그라운드로 실행시켜둔 채로 ./ex1 파일을 실행시킬 수 있었다. 이 때 실행되는 파일의 PID 는 3496 임을 확인하였다.

16) Kill it with "kill" command.

```
-bash-4.2$ kill 3496
```

이 후 위에서 파악한 PID 를 통해 실행중인 ex1 을 kill 명령어를 활용하여 중지시켰다.

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.

```
-bash-4.2$ ps
  PID TTY          TIME CMD
 3356 pts/6    00:00:00 bash
 3509 pts/6    00:00:00 ps
[1]+  Terminated              ./ex1
```

ps 를 통해 프로세스 정보들을 확인한 결과 기존에 진행되고 있던 PID 3496 이 없어진 것을 확인하였다.

18) Use "objdump -D -M intel ex1" to dump the assembly code of ex1.c. Find <main>.

```
00000000004005bc <main>:
  4005bc:       55                      push   rbp
  4005bd:       48 89 e5                mov    rbp,rsp
  4005c0:       bf 90 06 40 00          mov    edi,0x400690
  4005c5:       b8 00 00 00 00          mov    eax,0x0
  4005ca:       e8 c1 fe ff ff          call   400490 <printf@plt>
  4005cf:       48 8b 05 6a 0a 20 00    mov    rax,QWORD PTR [rip+0x200a6a]
   # 601040 <__TMC_END__>
  4005d6:       48 89 c7                mov    rdi,rax
  4005d9:       e8 e2 fe ff ff          call   4004c0 <fflush@plt>
  4005de:       eb fe                   jmp    4005de <main+0x22>
```

(출력 결과가 너무 길어 일부 생략하여 첨부하였습니다.)
objdump 명령어는 주어진 실행파일 안에 있는 기계어를 어셈블리어로 보여주는 명령어이다. objdump -D -M intel ex1 를 통해 ex1 파일의 기계어를 어셈블리어로

보여주고 –DM 옵션 중 –D 는 기계어를 어셈블리어로, -M 은 어셈블리어를 인텔문법으로 바꿔준다.

```
-bash-4.2$ objdump -D -M intel ex1

ex1:     file format elf64-x86-64


Disassembly of section .interp:

0000000000400238 <.interp>:
  400238:       2f                      (bad)
  400239:       6c                      ins    BYTE PTR es:[rdi],dx
  40023a:       69 62 36 34 2f 6c 64    imul   esp,DWORD PTR [rdx+0x36],0x646c2f
34
  400241:       2d 6c 69 6e 75          sub    eax,0x756e696c
  400246:       78 2d                   js     400275 <_init-0x1eb>
  400248:       78 38                   js     400282 <_init-0x1de>
  40024a:       36                      ss
  40024b:       2d 36 34 2e 73          sub    eax,0x732e3436
  400250:       6f                      outs   dx,DWORD PTR ds:[rsi]
  400251:       2e 32 00                xor    al,BYTE PTR cs:[rax]

Disassembly of section .note.ABI-tag:

0000000000400254 <.note.ABI-tag>:
```

<main> 부분을 찾아 objdump 를 실행한 결과를 보면 다음과 같다.

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

```
-bash-4.2$ vi ex1.c
```

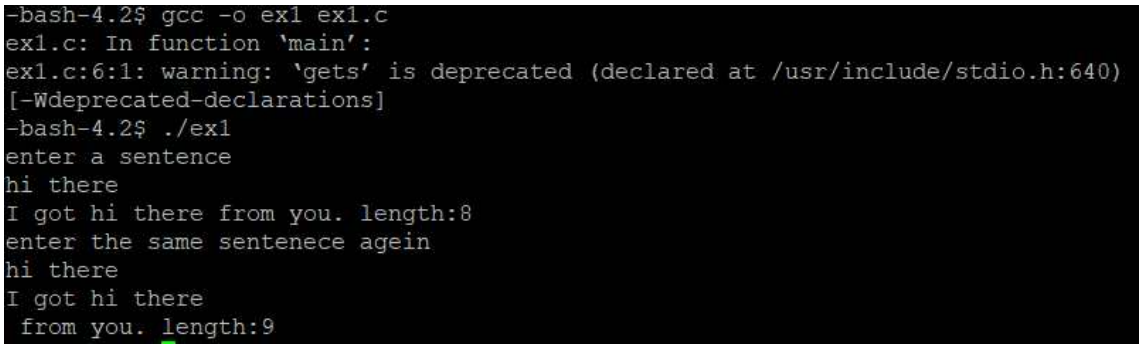
ex1.c 파일을 vi 편집기를 활용하여 open 한다.

```
#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 sentenece agein\n");
fgets(buf, 20, stdin);
printf("I got %s from you. length:%d\n", 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");
```

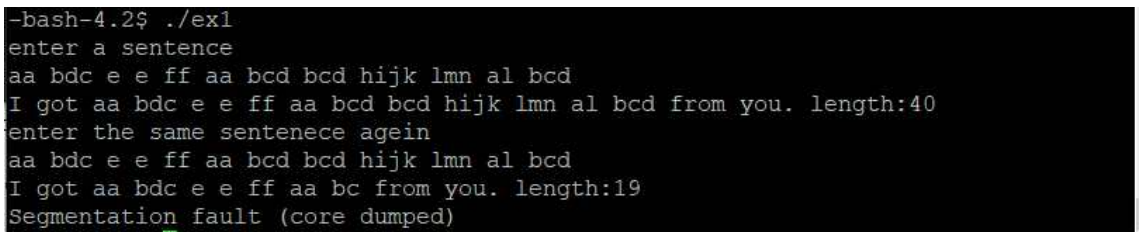
```
    fgets(buf, 20, stdin);
    printf("I got %s from you. length:%d₩n", buf, strlen(buf));
}
```

에 해당하는 코드를 'i'키를 통해 insert 모드로 변경하여 코드를 작성한다. 이후 esc 버튼과 :wq 를 통해 저장하고 파일을 빠져나간다.



이후 gcc –o ex1 ex1.c 를 통해 ex1.c 의 코드를 변환한 기계어를 ex1 파일에 저장한다. 이때 warning 은 fgets, gets 함수 사용으로 인해 발생한다.

./ex1 으로 ex1 파일을 실행시키면 앞서 작성했던 코드가 실행된다. 문장을 입력하면 문장의 길이에 대해 출력하고 한 번 더 같은 문장을 입력하면 fgets 가 줄바꿈을 인식하여 길이가 1 추가된 모습을 확인하였다.



또한, 문제에서 제시된 문자열을 입력해보았다. gets 함수는 20 의 크기를 가진 buf 의 크기에 상관없이 32 개의 문자열을 모두 출력하였다. 하지만 fgets 함수는 종료하는 공간을 제외한 19 개의 문자만 받는 것을 확인하였다.

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

```
-bash-4.2$ man gets
GETS(3)                    Linux Programmer's Manual                    GETS(3)

NAME
       fgetc, fgets, getc, getchar, gets, ungetc - input of characters and
       strings

SYNOPSIS
       #include <stdio.h>

       int fgetc(FILE *stream);

       char *fgets(char *s, int size, FILE *stream);

       int getc(FILE *stream);

       int getchar(void);

       char *gets(char *s);
```

(출력 결과가 너무 길어 일부 생략하여 첨부하였습니다.)

```
-bash-4.2$ man fgets
GETS(3)                    Linux Programmer's Manual                    GETS(3)

NAME
       fgetc, fgets, getc, getchar, gets, ungetc - input of characters and
       strings

SYNOPSIS
       #include <stdio.h>

       int fgetc(FILE *stream);

       char *fgets(char *s, int size, FILE *stream);

       int getc(FILE *stream);

       int getchar(void);

       char *gets(char *s);
```

(출력 결과가 너무 길어 일부 생략하여 첨부하였습니다.)

man gets 와 man fgets 를 통해 해당 함수의 manual 을 알 수 있었다. maunal 을 통해 gets 는 EOF 전까지, fgets 는 EOF 까지 받을 수 있다는 것을 확인하였다.

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

```
-bash-4.2$ vi ex1.c
```

vi ex1.c 를 입력하여 ex1.c 파일을 vi 편집기를 통해 열어준다.

command 모드에서 'i'키를 통해 insert 모드로 변경한다. 위와 같이 hello 를 출력하는 간단한 C 프로그래밍을 작성한다. 이후 esc 키를 누르고 :wq 를 통해 문자열을 저장하고 파일을 빠져나온다.

```
-bash-4.2$ xxd -l 20 ex1.c
0000000: 2369 6e63 6c75 6465 203c 7374 6469 6f2e  #include <stdio.
0000010: 683e 0a76                                h>.v
```

xxd ex1.c 는 ex1.c 에 저장된 값을 16 진수로 표현한다. 옵션 –l 20 을 통해 첫 20 바이트까지만 출력하였다. 사용자 편의를 위해 제공된 파트를 확인하면 '#include <stdio.h> v'까지 20 바이트임을 알 수 있다.

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

```
-bash-4.2$ xxd ex1
0000000: 7f45 4c46 0201 0100 0000 0000 0000 0000  .ELF............
0000010: 0200 3e00 0100 0000 a005 4000 0000 0000  ..>.......@.....
0000020: 4000 0000 0000 0000 8811 0000 0000 0000  @...............
0000030: 0000 0000 4000 3800 0900 4000 1e00 1b00  ....@.8...@.....
0000040: 0600 0000 0500 0000 4000 0000 0000 0000  ........@.......
0000050: 4000 4000 0000 0000 4000 4000 0000 0000  @.@.....@.@.....
0000060: f801 0000 0000 0000 f801 0000 0000 0000  ................
0000070: 0800 0000 0000 0000 0300 0000 0400 0000  ................
0000080: 3802 0000 0000 0000 3802 4000 0000 0000  8.......8.@.....
0000090: 3802 4000 0000 0000 1c00 0000 0000 0000  8.@.............
00000a0: 1c00 0000 0000 0000 0100 0000 0000 0000  ................
00000b0: 0100 0000 0500 0000 0000 0000 0000 0000  ................
00000c0: 0000 4000 0000 0000 0000 4000 0000 0000  ..@.......@.....
00000d0: 2c09 0000 0000 0000 2c09 0000 0000 0000  ,.......,.......
00000e0: 0000 2000 0000 0000 0100 0000 0600 0000  .. ............
00000f0: 100e 0000 0000 0000 100e 6000 0000 0000  ..........`.....
0000100: 100e 6000 0000 0000 4402 0000 0000 0000  ..`.....D.......
0000110: 5802 0000 0000 0000 0000 2000 0000 0000  X.........  .....
0000120: 0200 0000 0600 0000 280e 0000 0000 0000  ........(.......
0000130: 280e 6000 0000 0000 280e 6000 0000 0000  (.`.....(.`.....
0000140: d001 0000 0000 0000 d001 0000 0000 0000  ................
0000150: 0800 0000 0000 0000 0400 0000 0400 0000  ................
0000160: 5402 0000 0000 0000 5402 4000 0000 0000  T.......T.@.....
0000170: 5402 4000 0000 0000 4400 0000 0000 0000  T.@.....D.......
0000180: 4400 0000 0000 0000 0400 0000 0000 0000  D...............
0000190: 50e5 7464 0400 0000 2008 0000 0000 0000  P.td.... .......
```

xxd ex1 을 통해 ex1 에 저장된 기계어 값을 16 진수로 출력하였다.
우측에 ELF 를 보면 해당하는 '7f45 4c46'이 ELF 의 매직 넘버임을 알 수 있다.
그 다음 1byte 는 ei_class 인데 '02'이므로 64bit 환경임을 의미한다.
그 다음 1byte 는 ei_data 인데 '01'이므로 little endian 을 의미한다.
그 다음 1byte 는 ei_version 인데, '01'이므로 elf 파일의 버전을 의미한다.

그 다음 1byte 는 ei_osabi 인데, '00'이므로 시스템 v 를 의미한다.
나머지 7byte 는 ei_pad 인데, '00 00 00 00 00 00 00' 이다.
여기까지가 첫 line 16byte e_ident 의 구성이고
그 다음 줄의 시작의 2byte 는 e_type 인데, '02 00'이므로 ET_EXEC 파일임을 의미한다.
그 다음 2byte 는 e_machine 인데, '3e 00'이므로 amd64 임을 의미한다.