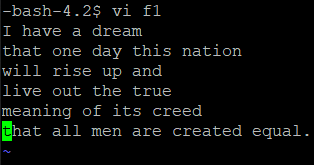
**System Programming Lect6 HW**

**001분반 / 12161756 / 윤성호**

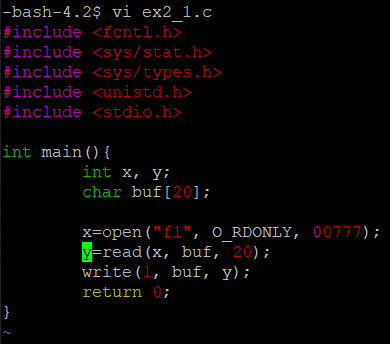
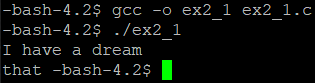
9. Homework

1) Make a file, "f1", and fill it with more than 20 bytes.



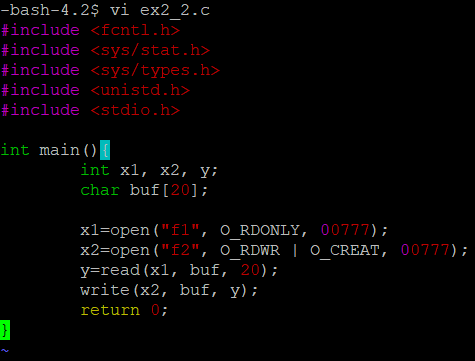
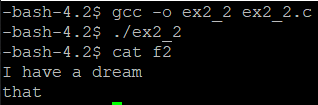
**= vi로 제시된 문장을 f1에 작성하였다.**

2) Try the code in 6-0), 6-1), 6-2), 6-3).

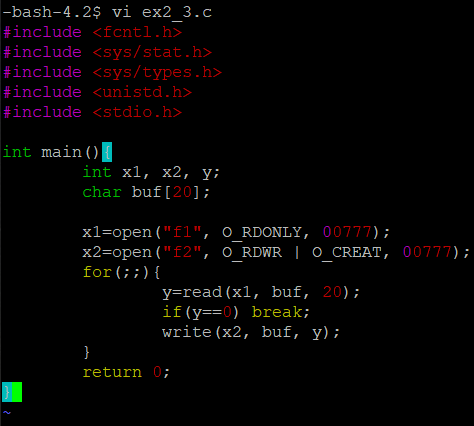
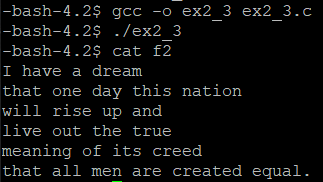
**= open 명령어로 f1 파일을 열고 read로 최대 20byte까지 읽은 후 write로 y bytes만큼 화면에 출력한다.**

**= 결과적으로 이 코드는 f1파일을 20bytes까지 출력하는 코드이다.**

**= f1 파일을 읽기 전용으로 열고 f2파일은 read and write로 연다. 만약 f2파일이 없다면 생성한다. 그 후 f1파일을 최대 20 bytes까지 읽어 buf에 저장한 후 buf의 내용을 f2에 저장한다. 바로 위 코드와는 달리 이 코드에서는 enter 입력 뒤로도 복사가 되었다.**

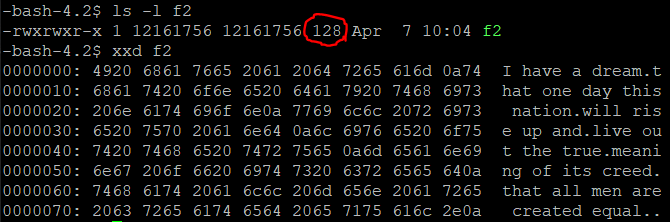
**= f1의 20bytes 내용을 f2에 복사하는 코드이다.**

**= f1은 읽기전용, f2는 읽고쓰기로 열고(없으면 생성) f1의 내용을 앞에서부터 20bytes씩 읽어 buf에 저장하고 buf에 저장된 내용을 f2 파일에 기록한다. y가 0이 되면(f1파일을 끝까지 읽었을 때) for문을 탈출하고 코드는 종료된다. f1의 내용이 f2에 모두 복사되었다.**

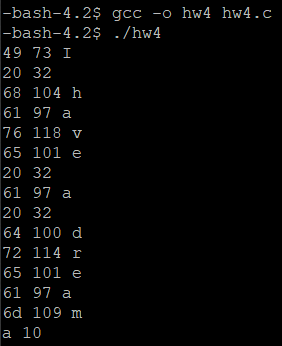
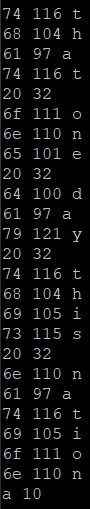
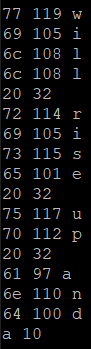
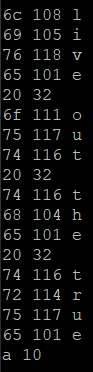
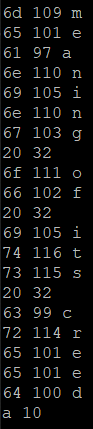
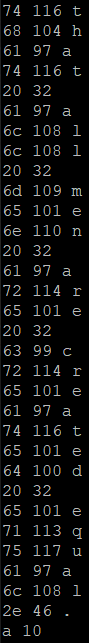
**= f1의 모든 내용을 f2에 복사하는 코드이다.**

3) Find the byte size of f2 with “ls –l f2”. Use xxd to find out the actual data stored in f2.



**= [ls -l f2]로 확인한 f2의 파일 크기는 128bytes였다. xxd로 확인한 값 역시 128bytes(8\*16[행\*열])로 동일하였다.**

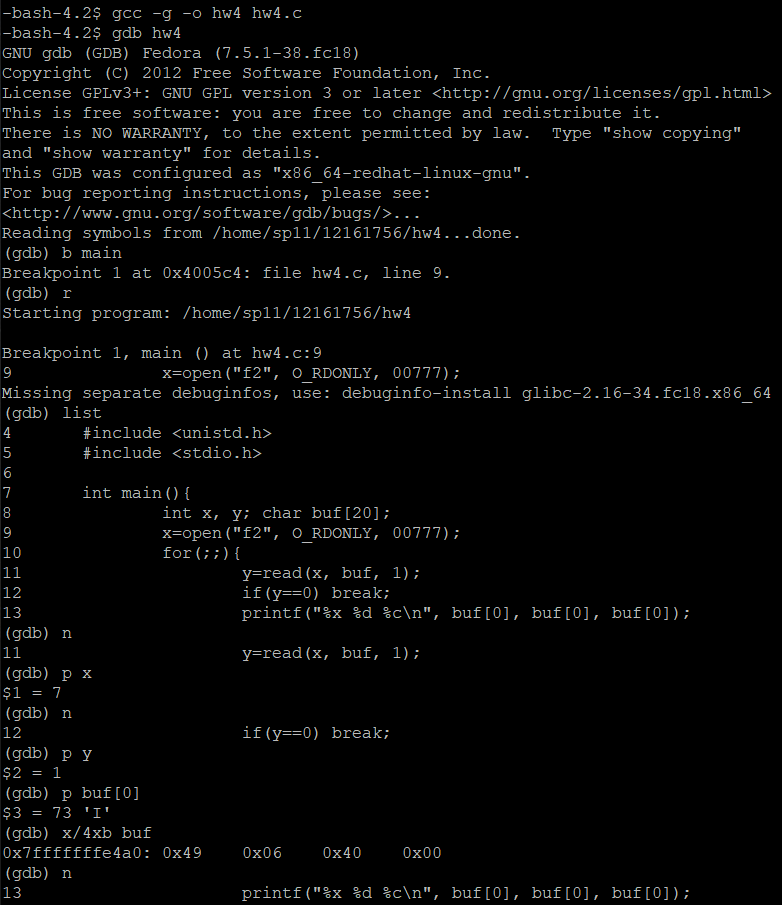
4) Write a program "hw4.c" that opens f2 and shows each byte of it in hexadecimal number, decimal number, and character. Use printf("%x %d %c\n", ...............) to display a number in various format.

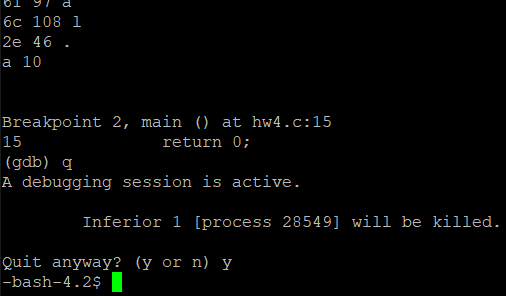
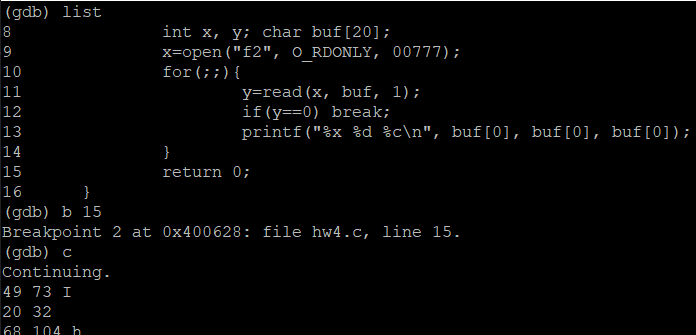
     

**= f2파일을 읽기전용으로 연 후 한 글자씩 buf에 저장한다. buf에 저장된 값을 16진수와 10진수, 문자값으로 출력한다. f2 파일을 모두 읽으면 for문을 탈출하고 코드가 종료된다.**

**space와 enter 입력을 포함한 f2의 내용이 한 글자씩 모두 출력되었다.**

5) Compile hw4.c with –g option and run gdb to execute each instruction one by one. Use “p” or “x” to check the value of a variable.





**= [gcc -g -o]로 hw4.c를 컴파일 한 후 [gdb]로 디버깅한다.**

**= [b main]은 main에 breakpoint를 설정한다. 9번째 줄에 breakpoint 1이 설정되었다.**

**= [r]은 프로그램을 시작한다. 시작하는 프로그램의 절대경로가 출력된 후 breakpoint에서 멈춘 모습이다.**

**= [list]는 code 목록을 출력한다. 코드 중 일부가 출력되었다.**

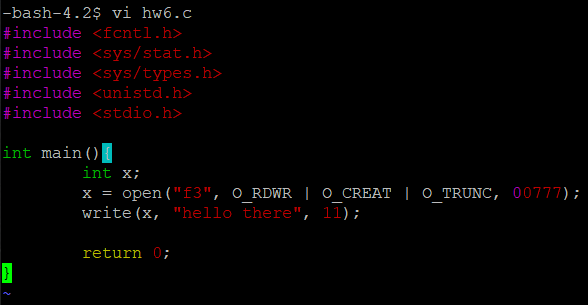
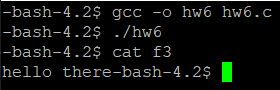
**= 이후 다음 줄을 실행하는 [n]과 변수값을 출력하는 [p 변수명]을 이용한다.**

**= 11번째 줄의 x에서는 오픈된 f2의 파일 번호가 7인 것을, 12번째 줄의 y에는 읽은 byte가 1byte임을, buf[0]에는 화면에 출력될 값이 'I'(73)인 것을 알 수 있다.**

**= [x/4xb buf]로 buf의 주소와 배열의 4bytes까지 16진수로 출력한다. 첫 1byte의 값이 0x49로 출력되어 이를 10진수로 변환하면 73, 아스키코드로 해석하면 ‘I’인 것을 알 수 있다.**

**= [b 15]로 루프 이후의 15번 줄로 break point를 이동하고 [q]로 디버깅을 종료한다.**

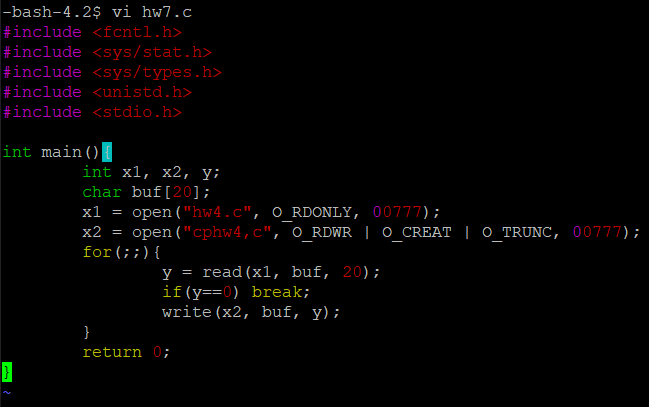
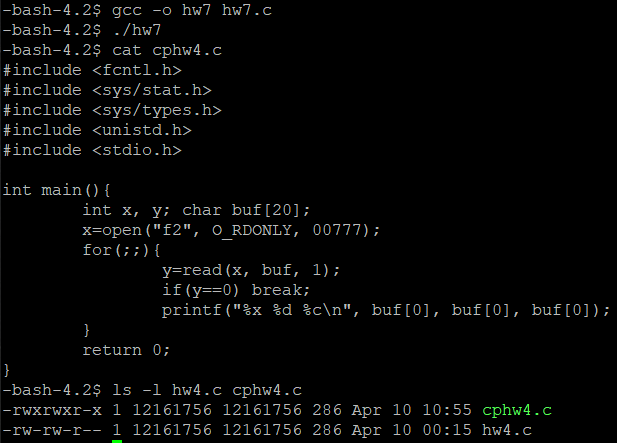
6) Write a program that creates a file and writes “hello there”in it. Use open() and write(). Confirm the result with "cat".

**= f3 파일을 연다. 만약 없으면 생성하고 이미 존재하면 O\_TRUNC로 인해 파일 내용을 비우고 새로 시작한다. 11바이트 문자열 ‘hello there’를 f3파일에 기록한다.**

**= 컴파일과 실행 후에 cat를 이용해 f3 파일을 확인해보니 ‘hello there’이 출력되는 것을 확인할 수 있다.**

7) Write a program that makes a copy for file "hw4.c" into another file "cphw4.c". Use open(), read(),and write(). Confirm that they are same with "cat" and "ls -l".

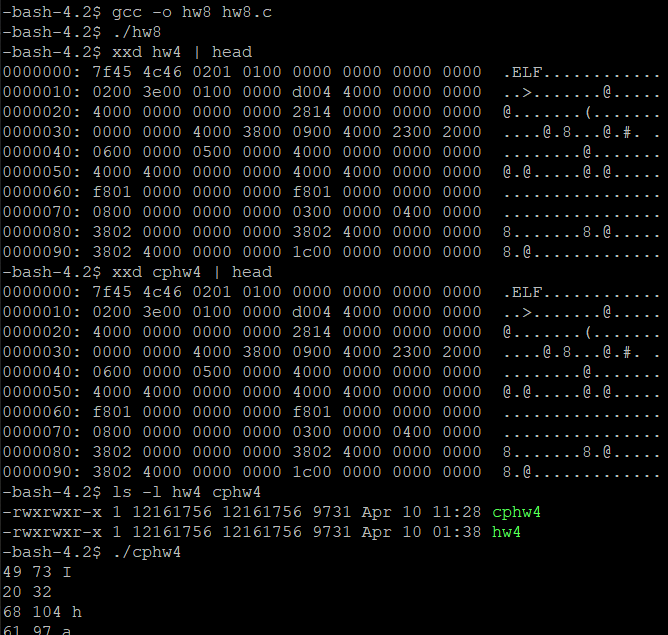
 

**= 이미 작성되있는 ‘hw4.c’ 파일은 읽기 전용으로, ‘cphw4.c’ 파일은 read and write로 연다. 이후 무한 루프를 돌면서 hw4.c 파일의 내용을 20bytes씩 읽고 y바이트 만큼 buf에 기록한다. 만약 hw4.c 파일을 끝까지 다 읽게되면 (y가 0인 상태) break로 루프를 탈출하고 코드가 종료된다.**

**= cphw4.c를 [cat]로 확인했더니 hw4.c의 내용이 모두 출력되었다. [ls -l]에서는 cphw4.c와 hw4.c의 용량이 286bytes로 동일한 것과 생성된 cphw4.c의 user 권한이 read and write를 포함하는 것을 확인할 수 있다.**

8) Write a program that makes a copy for file "hw4" (the executable file for "hw4.c) into another file cphw4. Confirm that they are same with "xxd" and "ls -l".

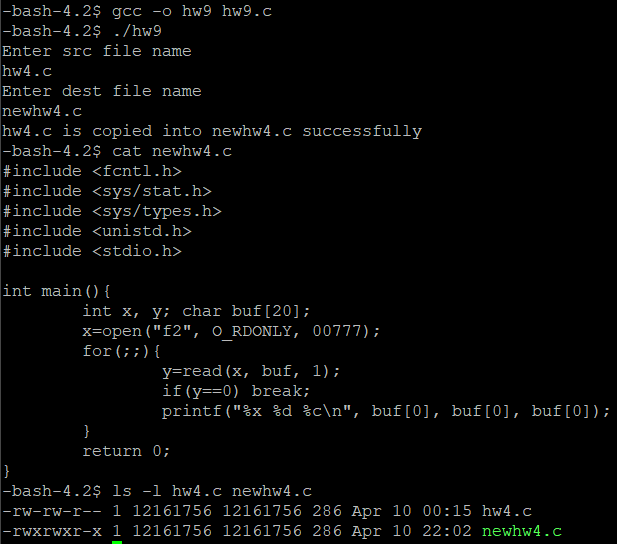
Execute cphw4 to see if it runs ok.

**= hw4 파일을 읽기전용으로, cphw4 파일은 read and write로 없다면 새로 생성하여 연다. 무한 루프를 돌면서 hw4 파일을 20bytes씩 계속 읽으면서 cphw4 파일에 기록한다. 다 읽으면(y=0) 루프를 탈출하고 코드는 종료된다.**

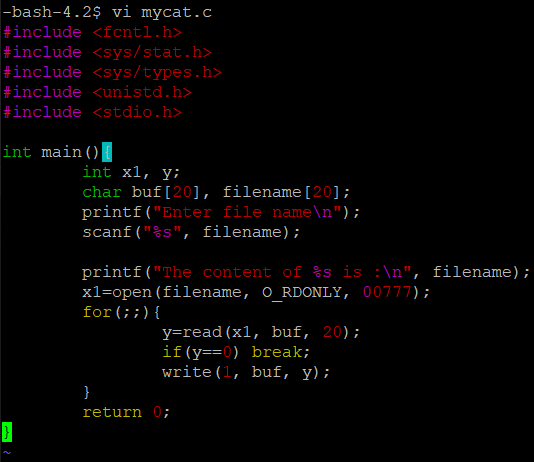
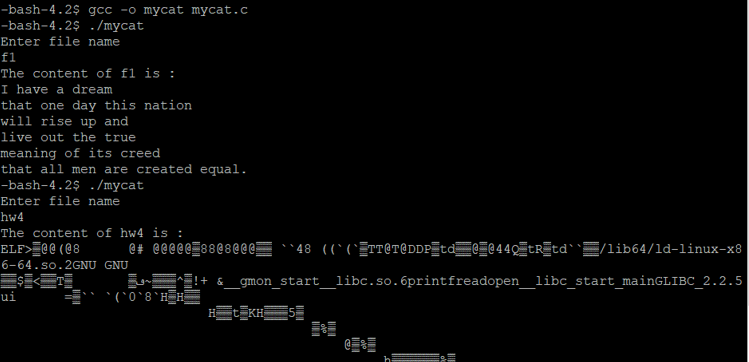
**= 컴파일 후 hw8를 실행하고 xxd와 head를 이용해 상위 10줄까지 hw4와 cphw4의 내용이 같은 것을 확인하였다. ls -l로도 cphw4와 hw4의 용량이 9731bytes로 동일한 것을 확인할 수 있다. 실제로 cphw4를 실행하자 hw4의 실행결과와 동일하게 출력되었다.**

9) Repeat 7). But get the name of the files from the user. Confirm that the result of copy with "cat" and "ls -l".



**= scanf로 복사의 대상이 되는 src 파일명과 복사 결과인 dest 파일명을 입력받은 후 open 명령어의 인자로 전달한다. 그 후 무한 루프를 돌면서 src 파일 내용을 20bytes씩 읽고 dest 파일에 기록한다. src 파일을 모두 읽으면 복사완료 메시지를 출력한 후 루프를 탈출하며 코드는 종료된다.**

10) Write "mycat" that displays the contents of a user-input file in the terminal in characters. Give a text file and a non-text file to mycat and explain the difference.

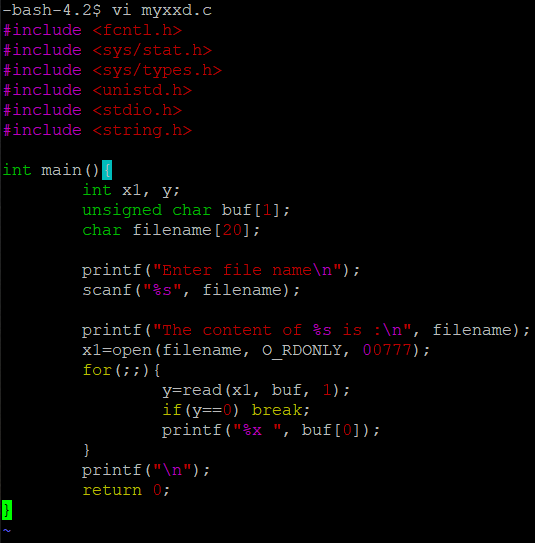
**= scanf로 filename을 입력받는다. 입력받은 파일을 읽기전용으로 연뒤 무한루프를 이용해 20bytes씩 buf에 저장하고 [write(1, buf, y)]를 이용해 buf에 저장된 내용을 y바이트씩 화면에 출력한다. 파일을 모두 읽으면 루프를 탈출하고 코드는 종료된다.**

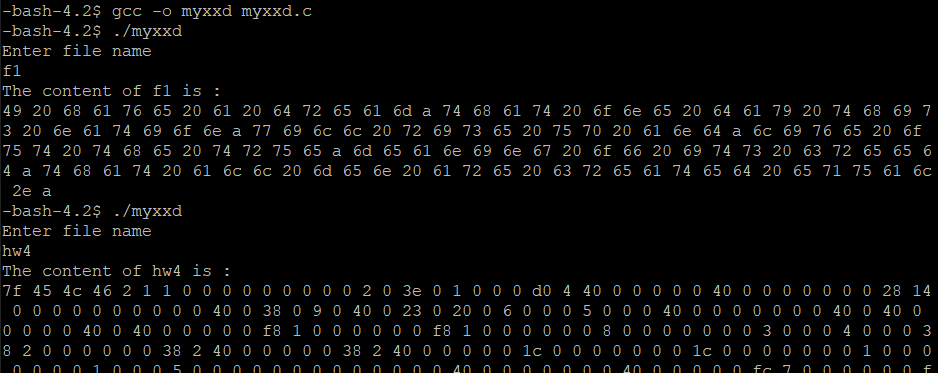
**= 코드를 컴파일 후 실행한 후 hw9-1)에서 직접 vi로 타이핑하여 작성한 파일인 f1을 입력하자 f1에 입력된 텍스트들이 모두 출력되었다.**

**= 반면, hw4.c 파일을 컴파일하여 생성한 실행파일 hw4를 입력하자 내용이 깨져서 출력되었다.**

**실행파일에는 기계어가 포함되어 있기에 아스키코드로 해석해 출력하는 [write(1,,)] 명령어로 인해 내용이 깨져 출력되는 것이다.**

11) Write "myxxd" that displays the contents of a user-input file in the terminal in hexadecimal numbers. Give a text file and a non-text file to myxxd and explain the difference. You need to use printf(“%x “, buf[i]) to display a byte in a hexadecimal number. Also declare the buffer as an array of unsigned char.

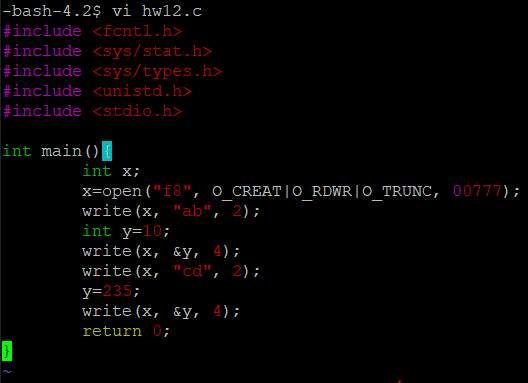
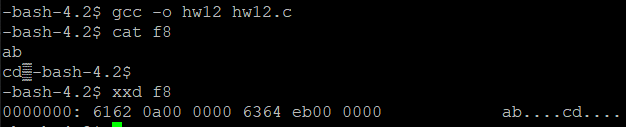




**= scanf로 파일명을 입력받고 해당 파일을 읽기전용으로 연다. 이후 무한루프로 파일 내용을 1바이트씩 buf에 저장 후 [printf(“%x ”,buf[0],)]로 저장 내용을 16진수로 출력한다. 파일을 다 읽으면 루프를 탈출하고 코드가 종료된다.**

**= 이번에는 f1와 hw4파일 모두 16진수로 잘 출력되었다. 문자나 기계어는 모두 16진수로 표현이 가능하기 때문에 출력에 문제가 없었다.**

12) Run following code and display f8 with cat and xxd respectively. Explain the results.

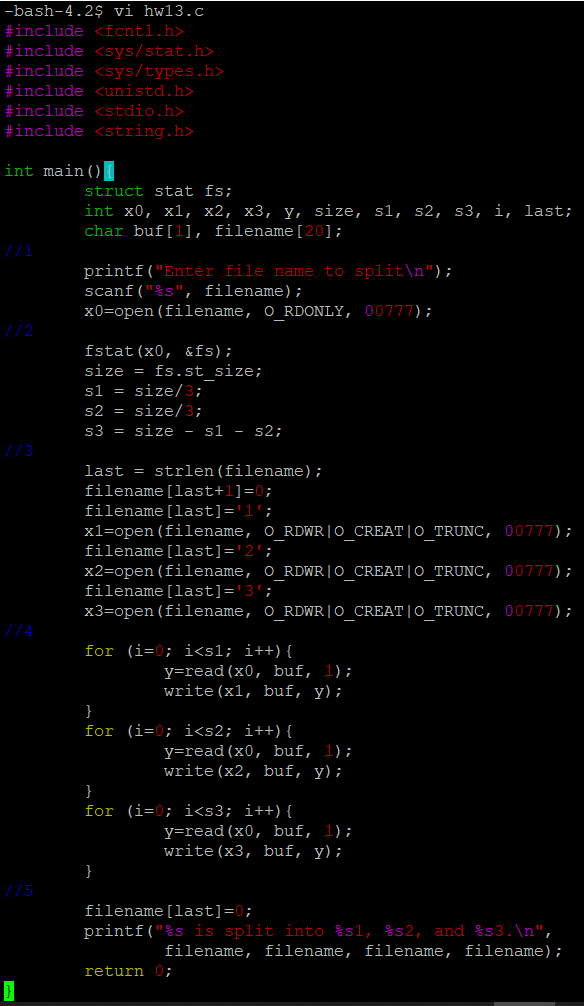
**= f8파일을 생성 후 읽기전용으로 연다. ‘ab’ 문자를 2바이트만큼, 10이 저장된 y의 주소값을 4바이트만큼 f8에 기록한다. 그 후 ‘cd’ 문자와 235가 저장된 y의 주소값을 4바이트만큼 f8에 기록한다.**

**= cat로 확인하자 문자인 ab와 cd는 잘 출력되었으나 주소값은 출력되지 않았다. cat 명령어는 파일을 아스키코드로 해석하여 출력하기 때문이다.**

**= xxd로 확인하자 ab와 cd는 6162, 6364로 출력되었고 이는 아스키코드 표에서 확인했을 때 각각 a, b, c, d를 나타내는 16진수 값이다.**

**= y의 주소는 각각 ‘0a00 0000’과 ‘eb00 0000’으로 출력되었다. 이는 intel cpu 등에서 사용되는little endian 표기법이다. 자리수가 작은 값부터 저장하며 반대로 big endian은 자리수가 큰 값부터 저장한다. big endian으로 위 y주소를 표기시 ‘0000 000a’, ‘0000 00eb’가 된다.**

13) Write a program that divides a given file into three small files of roughly equal size. Use fstat() to find out the size of a file.

**phase1 : fstat 함수를 사용하기 위해 [struct stat]로 파일 정보를 담는 구조체를 생성한다.**

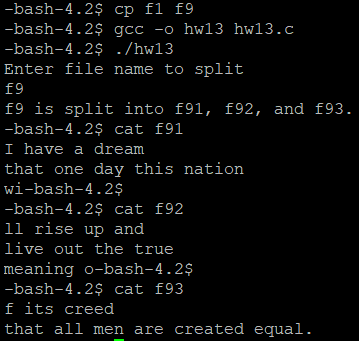
**phase2 : scanf로 파일명을 입력받고 그 파일을 읽기 전용으로 연다.**

**phase3 : fstat 함수에 파일 번호와 생성한 구조체의 주소값을 넘겨준다. 파일 크기인 [.st\_size]를 int형 size 변수에 담아 s1, s2, s3에 비슷하게 나눈다.**

**phase4 : 파일명의 길이를 last 변수에 저장하고 파일명의 last+1 배열에 0을 입력한다. 그 후 파일명 마지막 배열 바로 뒤 칸을 1, 2, 3로 바꿔가면서 파일을 연다.**

**phase5 : for구문을 이용해 파일의 내용을 앞에서부터 s1, s2, s3 바이트만큼 파일명1, 파일명2, 파일명3에 기록한다.**

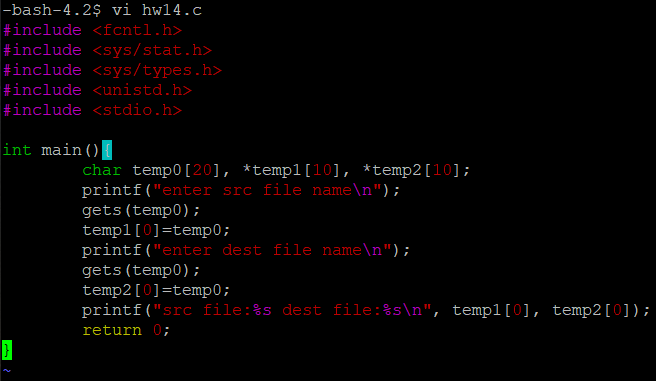
**phase6 : 파일명을 다시 원래대로 바꾸준 후 완료 메시지를 출력한다.**

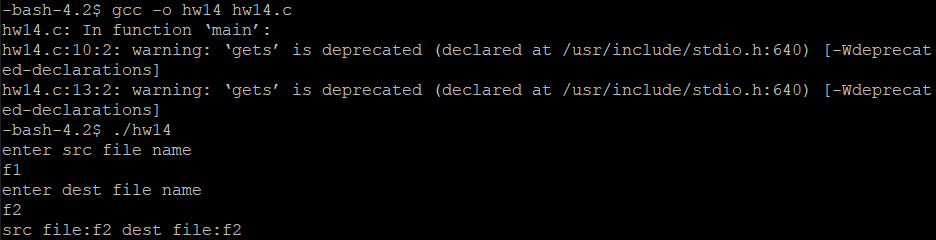


**= 파일 f1을 f9에 복사하고 hw13 프로그램을 이용해 f9파일의 내용을 f91과 f92, f93으로 분할한다.**

**분할 결과를 cat로 확인하였다.**

14) What is wrong with following program?

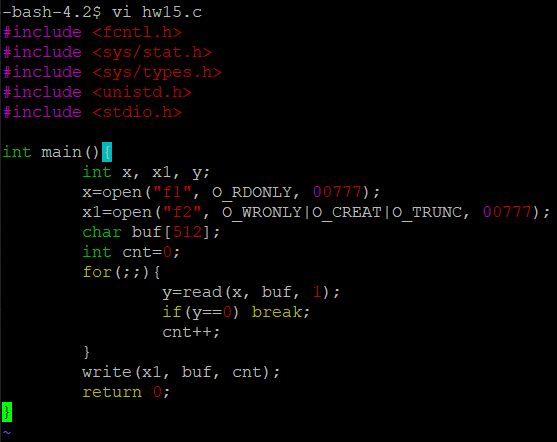
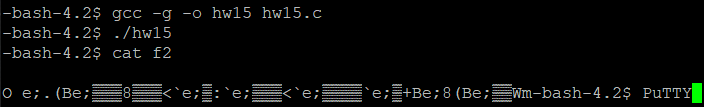




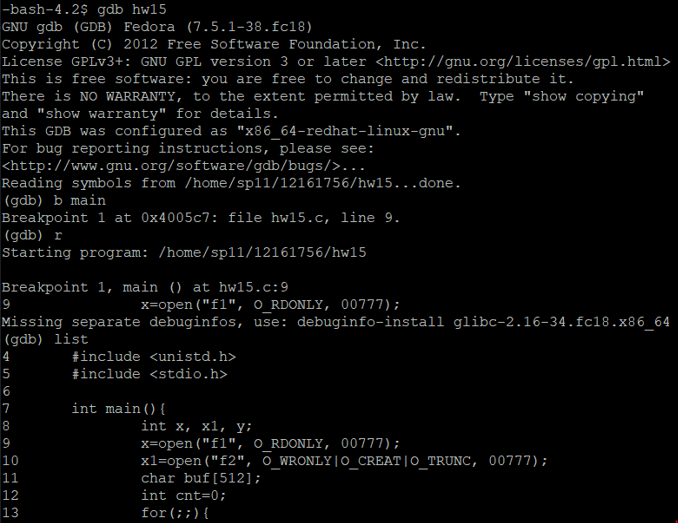
**= 위 코드는 gets로 char 배열 temp0을 입력받아 temp0의 주소를 포인터 배열 temp1[0]이 가리키도록한다. 그 후 다시 gets로 temp0을 입력받아 마찬가지로 temp0의 주소를 포인터 배열 temp2[0]이 가리키도록한다.**

**= 결국 포인터 배열 temp1[0]과 temp2[0]은 똑같이 temp0의 주소를 가리키기게 된다. 따라서, temp1[0]과 temp2[0]를 출력하면 마지막에 gets함수로 입력받은 dest 파일명이 동일하게 출력된다.**

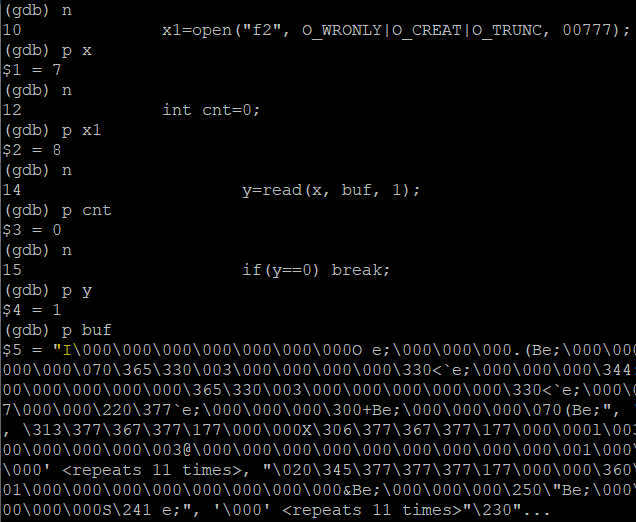
15) What is wrong with following program. Find the problem with GDB and fix it.

**= 제시된 코드를 실행하고 f2파일을 출력하자 내용이 깨져서 나오는 문제가 발생한다.**



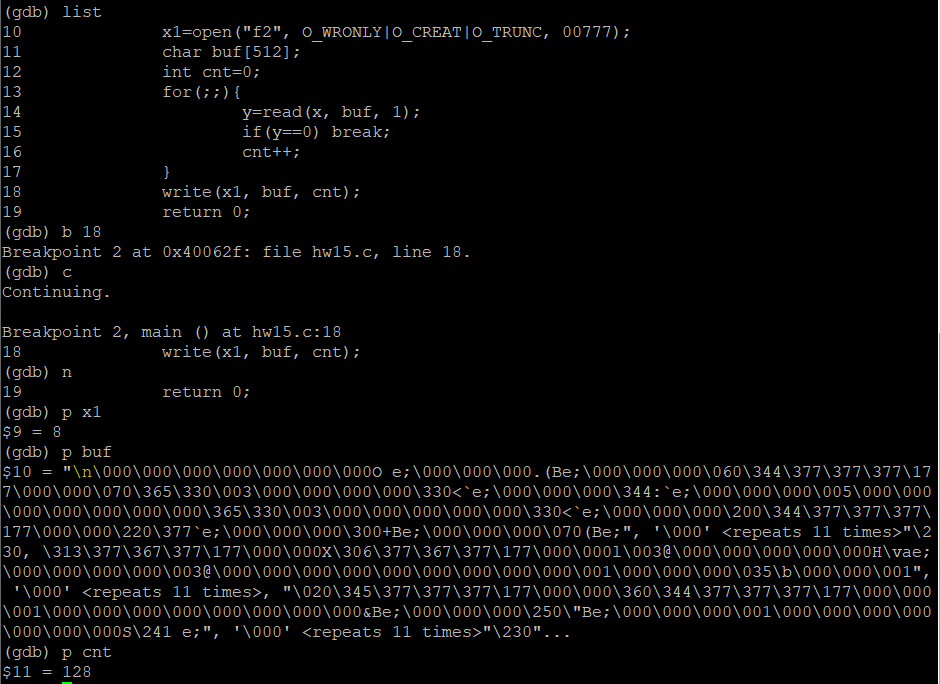
**= gdb로 디버깅을 진행하였다. break point를 main에 지정하고 프로그램을 실행한다. 이후 list로 코드 일부를 출력하였다.**



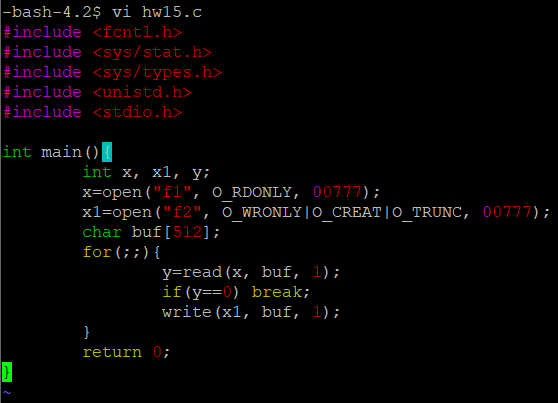
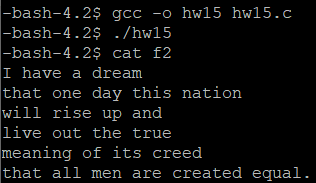
**= [n]으로 다음 줄을 실행해가면서 p를 이용해 x=7, x1=8, cnt=0, y=1인 것을 확인하였다. buf의첫 칸에는 f1의 첫 문자인 ‘I’가 입력되었다.**



**= 이후로도 for루프를 돌면서 변수 값을 확인했으나 별 다른 문제를 발견하지 못했다.**



**= 그래서 다시 코드 리스트를 출력하여 break point를 loop 이후인 18번에 걸어주었고 write에 넘겨지는 buf값을 확인하였더니 첫칸에 널문자가 입력되고 그 뒤로는 쓰레기값이 입력된 것을 확인할 수 있었다. 따라서 위에서 f2파일을 열었을 때 출력된 이상한 값들이 바로 buf의 이 쓰레기 값임을 추측할 수 있다.**

**= cnt 변수를 아예 제거하고 write 함수를 루프 안으로 집어넣어 f1의 내용을 1byte씩 buf에 저장 후 곧 바로 f2에 기록할 수 있도록 수정하였다.**

**= 코드 수정 후 원래 의도했던 f1을 f2에 복사하는 기능이 잘 구현되었다.**