9. Homework

**12201898 박준형**

\*\* About Indentation \*\*

When you are writing a program, please do indentation. **No point for code with no indentation!**

void main(){

int x;

if (x < 100){

int y;

y=x;

}

===> (After indentation)

void main(){

int x;

if (x < 100){

int y;

y=x;

}

}

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

$vi f1

I have a dream

that one day this nation

will rise up and

live out the true

meaning of its creed

that all men are created equal.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

2) Try the code in 6-0), 6-1), 6-2), 6-3), 6-4), 6-5). For 6-3) explain the strange output.

6-0)

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

f1파일을 앞에서부터 20바이트만큼 읽기전용으로 읽은후 그 값을 출력하는 프로그램이다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이스케이프 문자까지 포함해 20바이트를 읽어 출력하였다.

6-1)

코드>

printf 대신에 write(1, buf, y)로 교체해주었다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

write(1, ~, ~)을 이용하면, 터미널에 출력하라는 의미가 되므로 printf 대신에 사용할 수 있다. 그러나 개행문자를 넣을 수 없으므로 위 이미지처럼 나오게 된다.

6-2)

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

20바이트씩 읽는 read 함수를 y==0이 될때까지 반복하면 f1의 모든 데이터를 읽을 수 있다. 그리고 이 데이터를 다시 터미널에 출력하고 있다.

6-3)

코드>

write함수에서 y가 20의 상수로 바뀌었다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

다음과 같이 f1파일에는 없는 이상한 문자열이 출력되었다. 이는 우리가 write함수를 사용할 때, 출력할 바이트의 개수를 y가 아닌 20으로 했기 때문에 f1파일의 메모리공간 이외에 다른 공간에서 나머지 바이트를 가져와 출력했기 때문에 나타나는 현상이다.

6-4)

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

f1 파일은 읽기전용으로, f2파일은 읽기쓰기모드, 없으면 만들고, 있으면 모든 데이터를 지우도록 하였다. 그리고 f1파일에서 20바이트를 읽어 f2에 이를 작성한다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

f2에 f1에서 20바이트를 읽은 후 작성된 데이터가 보인다.

6-5)

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

이번엔 f1의 모든 데이터를 읽어 f2에 작성하는 마치 복사를 하는 듯한 알고리즘이다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

f1의 모든 내용이 f2로 복사되었다.

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

텍스트, 폰트, 스크린샷, 그래픽이(가) 표시된 사진

자동 생성된 설명

131바이트의 공간을 차지하고 있음을 알 수 있다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

실제 어떤 값이 들어있는지 확인하였다.총 131바이트가 저장되어있다.

3-1) Write a program that counts the number of bytes in f2. Compare it with the output of “ls –l f2”.

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

결과>



마찬가지로 131개의 바이트가 저장되어있음을 알 수 있다. 따라서 ls -l f2로 찾아본 바이트수와 일치하는 것을 확인할 수 있다.

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.

int x, y; char buf[20];

x=open("f2", O\_RDONLY, 00777); // open f2 for reading

for(;;){

y=read(x, buf, 1); // read next byte

if (y==0) break; // we read all, exit the loop

printf("%x %d %c\n", buf[0], buf[0], buf[0]); // display

}

결과>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

[이하생략]

위와 같이 한 글자씩 16진수, 10진수, char형 데이터로 출력하며 각 글자가 어떤 값을 가지는지 볼 수 있다.

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. For m1 mac, use lldb instead of gdb.

$ gcc -g -o hw4 hw4.c

$ gdb hw4

gdb) b main -- stop at main

gdb) r -- run

............

9 x=open("f2", O\_RDONLY, 00777); -- next line to execute

gdb) list -- show code list

gdb) n -- execute current line

11 y=read(x, buf, 1); -- line 9 has been executed. next is line 11

gdb) p x -- show x

$1 = 7 -- f2 is now file number 7

gdb) n

..........

gdb) p y

$2 = 1 -- we have read 1 byte

gdb) p buf[0]

$4 = 73 'I' -- assume we have 'I' in buf[0]

gdb) x/4xb buf -- show 4 bytes at buf in hexadecimal num

0x7fffffffe470: 0x49 0x06 0x40 0x00 -- we have 0x49=73='I' in buf[0]

(for lldb, you need an address to use x command.

p &buf -- print the address of buf

0x0016fdff496

x/4xb 0x0016fdff496 -- show 4 bytes at addr 0x0016fdff496

)

gdb) n

............ –- repeat a few times

gdb) list -- show rest of code

gdb) b 15 -- break point at line 15 (after loop)

gdb) c -- continue to that break point

gdb) q -- stop gdb

실행결과>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

gdb를 사용해보았다.

6) Write a program that creates a file and writes “how are you doing?”in it. Use open() and write(). Confirm the result with "cat".

x = open("f3", O\_RDWR | O\_CREAT | O\_TRUNC, 00777); // create f3

write(x, "how are you doing?", 18); // write 18 bytes in f3

코드>

텍스트, 스크린샷, 폰트, 바이올렛색이(가) 표시된 사진

자동 생성된 설명

결과>



잘 작성이 된 것을 확인할 수 있다.

6-1) Repeat Problem 6 but pass a string variable to "write" this time.

x = open("f3", O\_RDWR | O\_CREAT | O\_TRUNC, 00777); // create f3

..................

write(x, y, strlen(y)); // y is a string variable that has "how ..." string

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

사용자에게 문자열을 입력받아, 이 값을 f3 파일에 쓰도록 하였다. 이때 write의 인수로 y문자열의 길이가 들어가므로 문자열의 길이에 따라 알맞은 바이트 수만큼 write 할 수 있을 것이다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

입력한 문자열 그대로 f3에 들어간 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".

x1 = open hw4.c as O\_RDONLY

x2 = open cphw4.c as O\_RDWR | O\_CREAT | O\_TRUNC

for(;;){

y = read max 20 bytes from x1 into buf

if y is 0, break

write y bytes of buf into x2

}

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

결과>

텍스트, 스크린샷, 메뉴, 폰트이(가) 표시된 사진

자동 생성된 설명

방금 hw4.c에 작성했던 프로그램이 그대로 복사되어 cphw4.c 파일에 저장이 되었다.

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.

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

복사된 프로그램이 잘 실행되는 것을 확인할 수 있다. xxd로 실행한 결과, 둘간의 차이는 없었고, 파일 크기도 동일하였다.

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

Enter src file name

hw4.c

Enter dest file name

newhw4.c

hw4.c is copied into newhw4.c successfully.

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

결과>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

잘 복사된 것을 확인할 수 있다.

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.

$./mycat

Enter file name

f1

The content of f1 is :

I have a dream

that one day this nation

will rise up and

live out the true

meaning of its creed

that all men are created equal.

$./mycat

Enter file name

hw4

.............

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

결과>

text file>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

non-text file>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

실행파일을 열어보면, 다음과 같이 나오는데 이것은 char형으로 바이너리코드를 저장하고, char형으로 출력하다 보니 나타나는 결과이다.

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. Compare the result from the result of xxd.

$./myxxd

Enter file name

f1

The content of f1 is :

49 20 68 61 ..........

$ xxd f1

..................

$./myxxd

Enter file name

hw4

.............

$ xxd hw4

...............

코드>

텍스트, 스크린샷, 폰트, 메뉴이(가) 표시된 사진

자동 생성된 설명

결과>

1. Difference between text file and a non-text file  
   text file>  
   텍스트, 스크린샷, 폰트이(가) 표시된 사진

   자동 생성된 설명  
   non-text file>  
   스크린샷, 패턴이(가) 표시된 사진

   자동 생성된 설명  
   텍스트파일의 경우, 각 문자에 해당하는 아스키코드가 들어있는 것을 확인할 수 있다. 텍스트파일이 아닌 실행파일을 열었을 경우, 각 위치에는 아스키코드의 값이 아닌 명령어들이 16진수로 적혀있다.
2. run xxd and compare  
   text file>  
   텍스트, 스크린샷, 폰트이(가) 표시된 사진

   자동 생성된 설명  
   non-text file>  
   텍스트, 스크린샷이(가) 표시된 사진

   자동 생성된 설명

xxd 명령어를 이용해 같은 파일을 확인해보면, 같은 16진수값을 출력함을 알 수 있다.

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

int x;

x=open("f8", O\_CREAT|O\_RDWR|O\_TRUNC, 00777);

write(x, "ab", 2);

int y=10;

write(x, &y, 4);

write(x, "cd", 2);

y=235;

write(x, &y, 4);

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

1. write(x, "ab", 2);   
   코드에 의해 2바이트 공간에 “ab”가 아스키코드 값으로 입력됨.
2. int y=10;  
   write(x, &y, 4);  
   코드에 의해 y의 주소값이 넘어갔으므로, y의 주소로 가서 4바이트를 읽게 된다. 이때, 정수 10은 16진수로 0000000a 인데 인텔 cpu는 이를 little endian으로 낮은 바이트를 낮은 주소에 저장하므로 0a000000이 되게 되어 저장이 되었다.
3. 다음으로 “cd”를 2바이트 출력하였으므로 아스키코드 값으로 입력이 되었다.
4. 235는 16진수로 000000eb이므로 little endian으로 저장된 것을 고려하면 eb000000이 저장됨을 알 수 있다.

이때 cat으로 출력을 해보면, 아스키코드에 해당하는 값은 문자로 잘 출력됨을 확인할 수 있지만, 정수 값들은 아스키코드가 문자에 대응하지 않기 때문에 이상한 기호로 출력됨을 알 수 있다.

12-1) Run following code and display f8 with cat and xxd respectively. Explain the results.

int x;

x=open("f8", O\_CREAT|O\_RDWR|O\_TRUNC, 00777);

write(x, "ab", 2);

int y=10;

write(x, &y, 8);

write(x, "cd", 2);

y=235;

write(x, &y, 8);

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

12번 문제와 동일하지만, 정수를 저장하는 코드에 정수형의 바이트인 4바이트를 넘는 주소까지 가서 값을 받아와 저장하므로 03000000이라는 쓰레기값도 함께 저장된 것을 확인할 수 있다.

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.

Enter file name to split

f9

f9 is split into f91, f92, and f93.

코드>

텍스트, 스크린샷, 바이올렛색, 보라색이(가) 표시된 사진

자동 생성된 설명

분할한 파일의 용량은 원본 파일의 크기에서 빼주고 이를 다시 나머지 분할할 파일의 개수(size-i)만큼 나눈 값만큼 크기를 만들어줍니다.

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명



우선 파일분할을 원하는 파일의 이름을 입력받고 해당 파일의 크기를 fstat함수를 이용해 stat구조체에 저장해놓는다. 그리고 outer loop는 size의 횟수만큼 반복문을 돌면서 파일을 분할한다. inner loop는 파일 크기를 size-i로 나눈 값보다 크거나 같아질때까지 반복하면서 반복이 끝나면 분할될 파일 크기를 전체 크기에서 빼줌으로서 left\_size를 계산한다. 그릭 다음 반복에서는 left\_size를 남은 분할파일의 개수인 size-i로 나누면서 파일분할을 계속 반복한다. 이때 read에서의 입력받는 바이트의 크기만큼 분할파일 간에 크기 차이가 나게 되므로 이를 1바이트로 줄여보았다.

추가적으로 정수형 변수 i를 문자열로 바꾸어 저장하고자 sprintf함수를 이용하였다. 또한 strcat함수를 이용해 입력받은 파일의 이름에 붙여넣어 이를 분할파일의 이름으로 하는데 strcat함수는 원본을 바꾸는 함수이므로 반복때마다 temp에 원본파일의 이름을 strcpy하여 사용하였다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[코드 실행 결과]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[ls -l로 확인한 파일의 크기 – 잘 나누어진 것을 확인할 수 있다.]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[파일의 정보를 cat으로 확인해보았다. 문제없이 저장됨을 확인하였다.]

13-1) Modify your code in Problem 13 so that the user can specify the number of small files.

Enter file name to split

f9

How many smaile files you want to split it into?

5

f9 is split into f91, f92, f93, f94, f95

코드>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

size를 직접 입력받습니다.

텍스트, 스크린샷, 메뉴이(가) 표시된 사진

자동 생성된 설명

size에 맞게 문자를 출력해줍니다.

결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[5개로 파일을 분할해보았다.]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[용량이 잘 나누어진 것을 확인할 수 있다.]

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

[데이터도 이상이 없음을 확인하였다.]

14) What is wrong with following program?

char temp0[20], \*temp1[10], \*temp2[10];

printf(“enter src file name\n”);

gets(temp0);

temp1[0]=temp0;

printf(“enter dest file name\n”);

gets(temp0);

temp2[0]=temp0;

printf(“src file:%s dest file:%s\n”, temp1[0], temp2[0]);

실행 결과>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

위 코드는 temp0로 src, dest 파일의 이름을 입력받으면서 temp1[0], temp2[0] 포인터가 모두 temp0의 주소를 가르키고 있어 마지막으로 dest값이 temp0에 저장된 상태로 temp1[0], temp2[0]을 출력하니 같은 문자열이 출력되게 된다. 아마도 이는 의도한 바가 아닐 것이다. 이를 고치기 위해서는 동적할당을 통해 메모리 공간은 temp1[0], temp2[0]에 할당한 다음 strcpy를 하면 될 것이다.

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

int x, x1, y;

x=open(“f1”, O\_RDONLY, 00777);

x1=open(“f2”, O\_WRONLY|O\_CREAT|O\_TRUNC,00777);

char buf[512];

int cnt=0;

for(;;){

y=read(x,buf,1);

if (y==0) break;

cnt++;

}

write(x1, buf, cnt);

gdb 실행결과>

텍스트, 스크린샷이(가) 표시된 사진

자동 생성된 설명

선언과 동시에 초기화되지 않은 buf는 쓰레기 값으로 초기화가 되어있다. 이때 read함수를 통해 x로부터 1바이트를 읽어오면 buf[0]에 그 값이 저장된다. 그리고 다음 반복에서 그 값이 다시 새로운 값에의해 덮어씌워지게 된다.

그렇기 때문에 cnt는 계속 더해져 f1파일의 바이트수와 동일하게 되겠지만, 정작 f2파일에 넣어줄 정보인 buf는 마지막으로 저장된 buf[0]값만 남아있고, 나머지는 쓰레기값이다. 따라서 f2에는 쓰레기값이 저장되게 된다. 또 이때 buf의 크기는 512바이트이므로 만약 f1의 크기가 이 바이트를 넘어가면 cnt는 이 바이트수보다 커지게 되므로 cnt의 의미또한 상실될 것이다.

그렇기 때문에 다음과 같이 고칠 수 있다.

1. 만약 의도했던 바가 512바이트만 f1으로부터 읽어 f2에 저장하는 것이라면, 다음과 같이 코드가 작성될 수 있다.  
   텍스트, 스크린샷, 폰트이(가) 표시된 사진

   자동 생성된 설명

cnt가 512에 도달하면, 멈춘다. 그리고 read를 할 때 buf[cnt]에 각각 1바이트씩 저장을 한다. 그리고 f2에 buf에 들어있는 값을 옮긴다.

실행결과>

다음과 같은 f1이 존재한다면,

스크린샷, 텍스트, 패턴, 라일락이(가) 표시된 사진

자동 생성된 설명

이때 실행결과 f2는

ㅇ텍스트, 스크린샷, 폰트, 정보이(가) 표시된 사진

자동 생성된 설명

다음과 같다. 용량을 서로 확인해보면,





으로 f2는 딱 512바이트만큼 f1으로부터 복사됨을 확인할 수 있다.

1. 만약 의도하는 바가 512바이트씩 f1을 복사하여 f1의 모든 것을 f2에 복사하는 것이라면, 다음과 같이 코드가 작성될 수 있다.

코드>

텍스트, 스크린샷, 폰트이(가) 표시된 사진

자동 생성된 설명

결과>

스크린샷, 패턴, 라일락이(가) 표시된 사진

자동 생성된 설명

512바이트씩 복사하여 f1을 완벽히 복사하였다.





다음과 같이 데이터 크기도 동일한 것을 확인할 수 있다.