

# 파일 입출력

2020년 12월 16일 수요일    오후 12:26

- C에서는 파일 다루기가 조금 귀찮음
  - C#처럼 파일 내용을 한 번에 읽어와서 문자열 배열로 반환하는 등의 좋은 함수가 없다
  - 파일 복사하는 함수도 없음
    - 각 운영체제에서 제공하는 함수를 사용하거나
    - 그 기능을 직접 스트림 읽기/쓰기 함수를 사용해서 구현해야 함
  - 따라서, C에서 파일 관련 연산은 다 이런식임
    1. 파일을 열어서 파일스트림을 가져옴
    2. 그 파일스트림을 사용해서 하고 싶은걸 함
    3. 그 파일을 닫아 줌

## ■ 파일 열기

```
#include <stdio.h>

#define LENGTH (1024)

FILE* stream;
char list[LENGTH];

stream = fopen("hello.txt", "r");

if (fgets(list, LENGTH, stream) != NULL) {
    printf("%s", list);
}
```

- 파일 닫기를 추가해줘야 하는 코드임
- 파일에서 한 줄 읽어오고 출력하는 코드
- fopen() 함수

```
FILE* fopen(const char* filename, const char* mode);
```

- filename으로 지정된 파일을 열음
- 열 때 사용하는 모드는 mode로 지정
  - 예: 읽기 전용, 이진 파일 등
- 반환값은 파일 스트림 포인터

### • 파일 열기 모드 (1)

모드	설명	파일이 이미 있다면	파일이 없다면
r	(read) 파일을 읽기 전용으로 연다	파일의 첫부분부터 읽는다	열기에 실패한다
w	(write) 파일을 쓰기 전용으로 생성한다	파일의 내용을 모두 없앤다	새 파일을 생성한다
a	(append) 파일에 이어 쓴다	파일의 끝부분부터 읽는다	새 파일을 생성한다
r+	(read extended) 읽기/쓰기용으로 파일을 연다	파일의 첫부분부터 읽는다	오류
w+	(write extended) 읽기/쓰기용으로 파일을 생성한다	파일의 내용을 모두 없앤다	새 파일을 생성한다
a+	(append extended) 읽기/쓰기용으로 파일을 연다	파일의 끝부분부터 읽는다	새 파일을 생성한다

- w+ 는 읽기/쓰기 전용인임. 근데 파일의 내용을 모두 없애는데 뭘 읽느냐?
  - 파일을 생성하고 작성하다가 지금까지 작성한 것을 다시 읽으려 하는 것
- 가장 많이 쓰는건 r, w, a 이정도

- 나머지는 필요할때 참고해서 사용할 것
  - r+와 w+차이는 확실히 알도록 하자

## • 파일 열기 모드 (2)

- b를 붙이면 이진 모드로 파일을 열
  - rb, wb, ab, r+b, w+b, a+b
- 이진 모드란
  - 유닉스 계열에서는 아무 차이가 없음
    - rb나 r이나 동일
  - 윈도우에서 새줄 문자 처리하는 것만 달라짐
    - 윈도우에서 새줄 문자는 \r\n인 반면 유닉스 계열은 \n임

## ② 윈도우에서 파일 열기

- 파일 읽기용(r, r+, rb, r+b)으로 열 때

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	48	65	6C	6C	6F	0D	0A									

test.txt

◆ 'H','e','l','l','o','\r','\n'

프로그램에서 텍스트 모드로 파일 열었을 때

H	e	l	l	o	\n				
---	---	---	---	---	----	--	--	--	--

파일 스트림 버퍼

\n으로 저장

프로그램에서 이진 모드로 파일 열었을 때

H	e	l	l	o	\r	\n			
---	---	---	---	---	----	----	--	--	--

파일 스트림 버퍼

그대로 읽음

- 파일 쓸 때

B	y	e		b	y	e	\r		
---	---	---	--	---	---	---	----	--	--

파일 스트림 버퍼

프로그램에서 텍스트 모드로 파일 열었을 때

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	42	79	65	20	62	79	65	0D	0A							

\r\n으로 씀

- ◆ 문자열에 "r\n"이 있었다면, 두 모드 다 "r\n"으로 씀

프로그램에서 이진 모드로 파일 열었을 때

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000000	42	79	65	20	62	79	65	0A								

그대로 씀

## ■ 파일에 쓰기/읽기

### • 파일 쓰기

```
#include <stdio.h>
#include <string.h>

#define LENGTH (5)
```

```
FILE* stream;
int scores[LENGTH] = { 100, 34, 95, 56, 72 };

stream = fopen(filename, "wb");

fwrite(scores, sizeof(scores[0]), LENGTH, stream);
```

```
write_file("hello.txt");
```

- "w", "w+"로 파일을 쓸 경우 원래 파일에 있던 내용이 모두 사라짐을 주의
- 위 코드는 바로 파일에 써지지 않음
  - 보통 쓰기는 버퍼링 때문에 바로 파일에 저장되지 않음
    - 1) '\n' 이 들어오거나

2) fflush( )를 호출해야 함

- fwrite( )는 '\n'을 '\n'으로 인식을 못 함

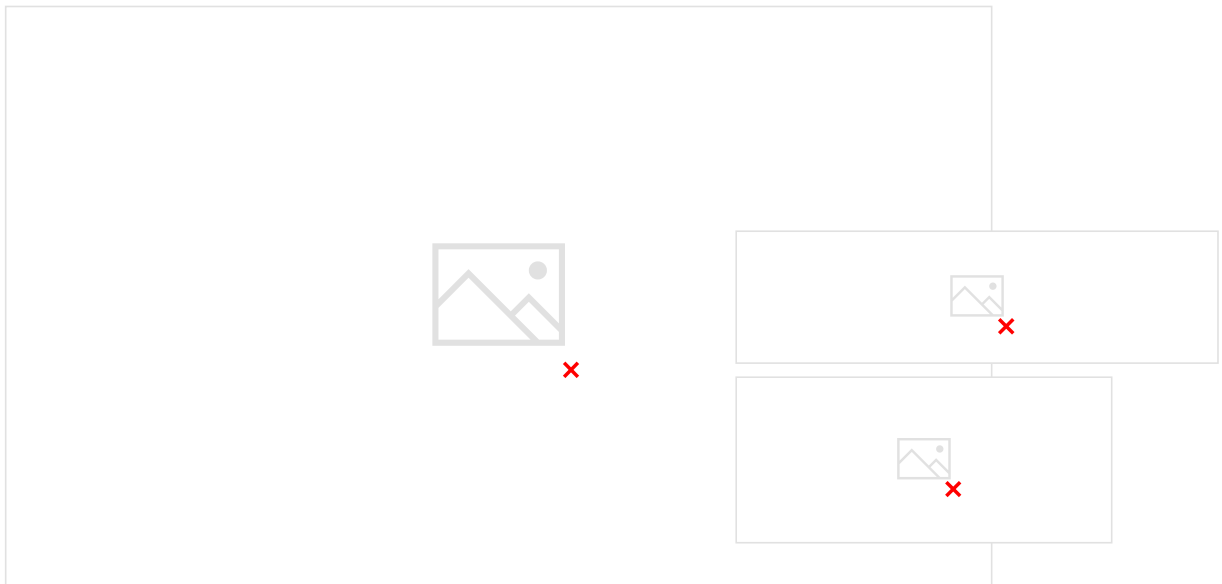
```
size_t fwrite(const void* buffer, size_t size, size_t count, FILE* stream);
```

- 첫 번째 인자로 받는 buffer가 char\*도 아니고 int\*도 아니고 그냥 void\*임
- 즉 fwrite( ) 입장에서는 그냥 비트 패턴이 줄줄이 들어옴
  - char\*로 들어오면 1바이트 단위로 읽어서 아스키코드로 인식하지만 void\*이기 때문에 의미 없어짐
  - 0x0A가 fwrite( ) 입장에서 'wn'을 의미하는 건지 정수 '10'을 의미하는 건지 알수가 없음
- 따라서 fflush( )만이 유일한 해결법
- 바로 파일에 쓰려면 fflush( ) 호출



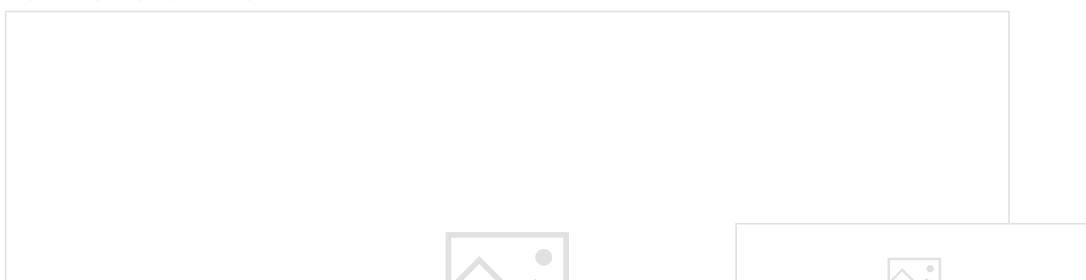
- 아직 파일 닫기를 하지 않은 미완성 코드임

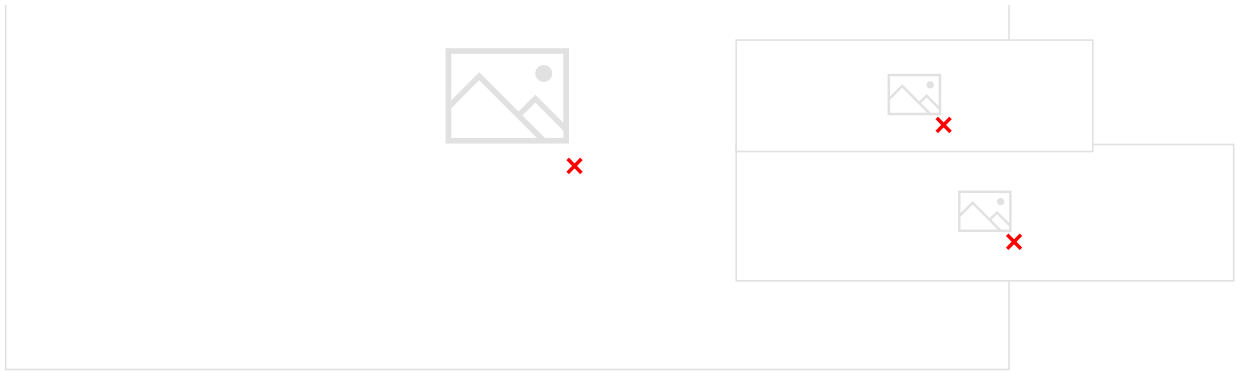
## • 파일 읽기



- 모드를 "rb" 말고 "r"로 해도 되지만. 둘의 차이는 새줄 문자 처리해주는 것 밖에 없음  
따라서 둘 중 아무거나 써도 무방
- LENGTH가 6이므로 5문자씩 읽음
  - 그래서 3줄로 출력이 된다
- 파일 위치 표시자 위치가 바뀜
  1. 처음에는 "H"에 있다가 "Hello" 출력 후
  2. " "로 표시자를 옮긴 후 " POCU" 출력 후
  3. "!"으로 이동 후 "!"을 출력한다.
- 파일 닫기를 하지 않은 미완성 코드

## • 파일에 이어 쓰기





- 원래 hello POCU!가 써있었고 콘솔에서 한 줄을 읽어서 Good night!를 씀
- fwrite( )의 두 번째, 세 번째 매개변수를 sizeof(data)와 strlen(data)를 쓸 수도 있다
  - char가 몇 개 저장됐는지 알고싶으면 i이렇게 하곤 됨
- 파일 위치 표시자는 파일을 처음 열었을 때 제일 뒤에 가있다.
- 파일 닫기를 하지 않은 미완성 코드

## ■ 파일 닫기

- 파일은 운영체제가 열어주는 것
- 운영체제는 우리가 언제 파일을 다 써서 필요없는지 모름
- 따라서 직접 말 안 해주면 알아서 닫아주지 않음
  - 계속 파일을 열기만 하면 어느 순간 운영체제가 더 이상 파일을 열 수가 없다며 뺄 수 있음
- C#에서는 파일을 알아서 닫아줬음

### • 파일 닫기



- 파일을 닫음
- 성공하면 0, 실패하면 EOF를 반환
- 버퍼링 중인 스트림은 이렇게 작동
  - 출력 스트림: 버퍼에 남아있는 데이터는 파일로 다 보냄
  - 입력 스트림: 무시하고 닫음
- 파일 닫기를 까먹는 경우
  - 보통 사람들은 파일을 열고 데이터를 처리하는 코드를 작성하다가 파일을 닫는 코드 작성하는 걸 까먹는다
  - 까먹지 않으려면,  
코드를 작성할 때 파일을 열고 바로 파일을 닫는 코드를 작성한 다음에 데이터를 처리하는 코드를 작성한다

## ■ 파일 오류처리

- 파일을 다룰 때 문제점
  - 파일이 없는 경우
  - 있었는데 누가 지우는 경우와 같은 통제 못하는 환경
- C#에서는 실패하면 예외처리를 했음
- 하지만 C에서는 예외처리가 없음
  - fopen( )함수는 실패하면 널 포인터를 반환



- 이것을 활용해서 파일 열기 실패 시 오류 메시지를 출력할 수 있음

### • 파일 열기 실패 시 오류 메시지 출력하기



×



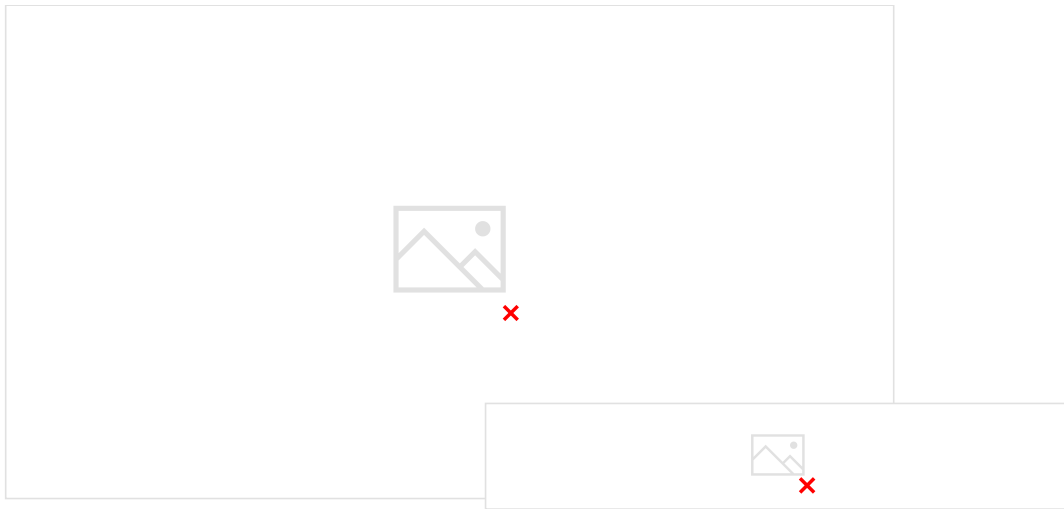
×

- `fclose( )`는 성공하면 0을 반환. 따라서 `fclose(stream) != 0` 이라 함
- `stdin`이 아닌 `stderr`에 출력했음

- **stderr란?**

- 프로그램 실행될 때 자동으로 3개의 스트림을 만들어줬다고 함
  - `stdout`
  - `stdin`
  - `stderr`
- `stderr`은 `stdout` 과 비슷. 둘다 출력해줌
  - `stdout`, `stderr` 모두 까만 화면에 같이 섞여나옴
  - 이 것을 분리하려면 입출력 리디렉션(`redirection`)을 배워야 함
- 다만, `stderr`는 오류 관련 메시지를 출력하는 전용 스트림
  - 반드시 규칙은 아니고 그냥 관례상
  - `stderr`은 버퍼링을 안 쓴다
    - 오류는 바로바로 보여줘야 하니깐

- 파일 열기에 실패할 경우




- if (!stream)
  - stream이 널 포인터이면 코드를 실행하라는 뜻
    - 널 포인터는 비트패턴이 0이므로 '!널포인터'를 하면 0이 아닌 수 '참'이된다.
  - if(stream) 이런식으로 코드를 쓸 때도 있는데
    - stream이 존재하면 코드를 실행하라 라는 뜻
- 실패한 이유를 알고 싶을 때
  - 그냥 실패했다단 사실 말고 왜 실패했는지 이유를 알고싶을 때는?
  - 몇몇 표준 라이브러리 함수들이 실패할 때 그 이유를 숫자(오류코드)로 어딘가에 저장해 둬
  - 그게 errno라는 매크로(#define)
    - <errno.h>에 있음
- 상세한 오류 정보를 보여주는 코드



- 오류를 말로 설명해주는 함수 strerr( )
  - <string.h>에 정의 됨
  - errno를 넣으면 문자열로 된 친절한 설명을 돌려줌
    - errno라는 매크로는 출력하면 그냥 숫자일 뿐임
    - 그것을 strerror( )라는 함수에 넣어주면 친절하게 오류를 문자열로 반환해줌
- 더 간단하게 오류 정보를 출력하려면? perror( )



×

- 
  - `fprintf(stderr, "%s - %s", filename, strerror(errno));`를 내부적으로 해주는 함수
- C에서의 오류처리는 보통 이런 식
  1. 함수가 곧바로 오류코드를 반환
  2. 내부적으로 오류코드를 전역 변수로 들고 있고 있다가 검사
    - `perror()`같은 걸로 오류 메시지를 불러 오면 됨
    - 하지만 까먹기 쉽다
    - 함수 문서를 안보면 이렇게 내부적으로 세팅되어 있는건지 잘 모른다.

## ■ 파일 탐색

- 스트림의 위치
  - 프로그래밍 입문에서 스트림은 한 방향으로 흐르며 읽는다고 했음
  - 근데 다음에 쓸 위치 또는 읽을 위치로 다른 곳으로 옮길 수 있을까?



×

- 스트림은 한 방향으로 흐르지만 다른 곳으로 이동해서 다시 흐르게 할 수 있을까?
    - 키보드 입력같은건 시간을 되돌릴 수 없으니 안되겠지만, 파일 스트림은 가능할 것
- 파일 위치 표시자(file position indicator)
  - 스트림 안에서 현재 위치를 나타내 주는 것



×

- 스트림의 총 3개의 표시자가 있음
  1. EOF 표시자
  2. 오류 표시자
  3. 파일 위치 표시자
    - 처음 2개는 `clearerr()`로 지울 수 있음
    - 파일 위치 표시자는 오류 상황을 나타내는 게 아니니 지우는 게 말이 안 됨

- 스트림 위치를 시작으로 돌리는 함수: `rewind()`

×

- 파일 위치 표시자를 처음으로 돌리기
- 이거 말고 다음에 나오는 함수를 대부분 씀
- `rewind()`를 사용해서 파일 두번 읽는 함수



×

- `num_read = fread(data, sizeof(data[0]), LENGTH, stream);`

- 스트림 위치를 임의의 위치로 옮기는 함수: `fseek()`

×

- 파일 위치 표시자를 `origin`으로부터 `offset`만큼 이동 함
  - `offset`은 어떤 기준점으로부터 얼마만큼 떨어져 있냐를 말함
- `origin`에는 세 종류의 매크로가 있음
  - `SEEK_SET`: 파일의 시작
  - `SEEK_CUR`: 현재 파일 위치
  - `SEEK_END`: 파일의 끝
    - ◆ 요즘 운영체제는 문자 1byte를 텍스트 파일에 저장하면 텍스트 파일의 크기가 1 바이트가 아니라 1KB가 기본인 경우가 많다.
    - ◆ 운영체제에 따라 `SEEK_END` 매크로를 호출하면 문자열의 끝(1byte)이 아닌 문자열이 저장 된 텍스트 파일의 블록(1KB)의 마지막으로 이동할 수도 있다
    - ◆ 그래서 `SEEK_END` 매크로는 강제하지 않고 운영체제에 따라 다를 수 있다
      - ◇ 우리가 보통 운영체제는 `SEEK_END`를 잘 지원해 준다
- 반환값
  - 위치 이동에 성공하면: 0을 반환
  - 위치 이동에 실패하면: 0이 아닌 수를 반환
- 파일의 앞에서 약간 뒤로 이동하는 코드





- SEEK\_SET에서 4바이트 만큼 이동하고 읽음(4A부터 읽음)



- 현재 위치에서 뒤로 돌아가기



- 이동에 실패한 경우



- 파일의 끝에서 앞으로 갈 수는 있음
  - 하지만 파일의 처음에서 앞으로는 안 됨



- 스트링의 마지막 위치에서 뒤로 한 칸 가는 것은 스트링을 확장했다고 하면 가능한 것
- 하지만, 처음 위치에서 앞으로 한 칸 더 가는건 말이 안됨

- 파일 위치 표시자의 현재 위치를 알 수 있는 함수: `ftell()`



- 파일 위치 표시자의 현재 위치를 알려주는 함수
- 실패하면 -1 반환
  - 실패는 일반적인 파일 스트림에서는 발생하지 않음
  - 파일 크기가 없는 파일 스트림에서 발생
    - 예: 소켓, 파이프 등등
  - 파일의 크기가 제공되지 않는 경우에도 발생 가능
    - 예: 가상 파일 시스템
- 바이너리 모드로 열었을 때는 파일의 시작 지점으로부터 몇 바이트 떨어져 있는지를 알려 줌
- 텍스트 모드로 열었을 때는 어떤 값을 반환할지 정해져 있지 않으나 유효한 값을 반환해줌
  - `fseek()`의 `offset` 매개변수로 사용할 때 유효한 값을 반환해 줌