
C 프로그래밍 및 실습

14. 파일 입출력

세종대학교

p. 40 그림 19 칸에서 20 칸으로 수정

p. 60 그림 수정

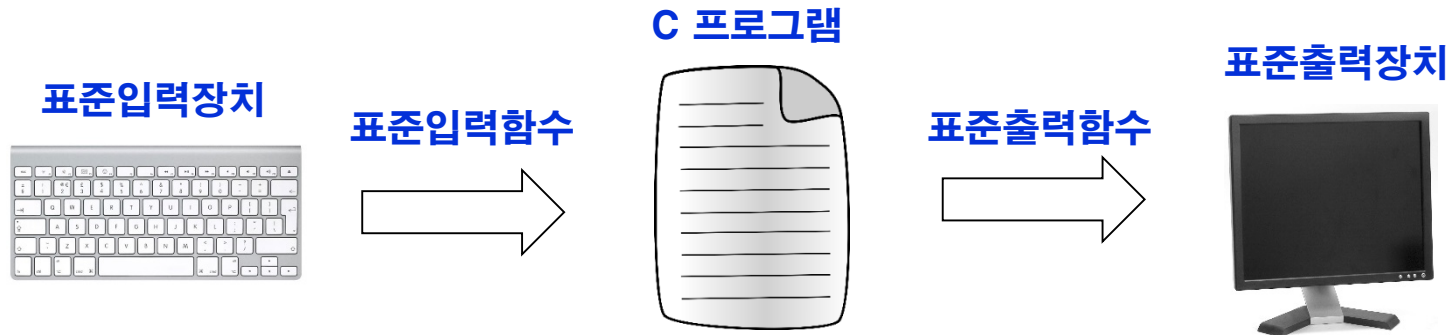
목차

- 1) 파일 입출력 개요
- 2) 파일 입출력 절차
- 3) 텍스트파일 vs. 이진파일
- 4) 텍스트파일의 입출력 함수
- 5) 이진파일의 입출력 함수 (심화 내용)
- 6) 기타 파일 입출력 관련 함수 (심화 내용)

1) 파일 입출력 개요

■ 표준 입출력

- 표준입력장치(키보드)를 통해 입력 받아 처리하여 표준출력장치(모니터)를 통해 결과를 보여주는 것
- 표준 입력 함수: scanf(), getchar(), gets()
- 표준 출력 함수: printf(), putchar(), puts()
- 프로그램이 종료되면 입출력의 결과는 사라짐



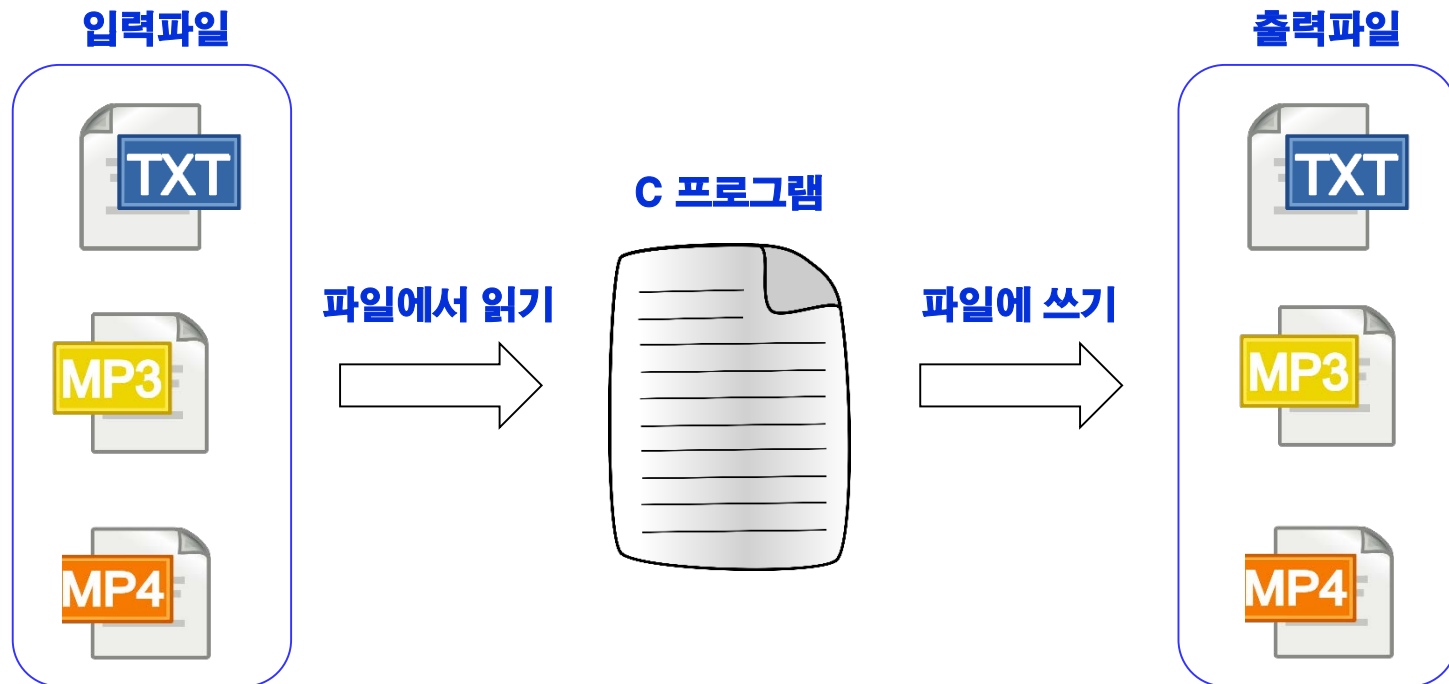
1) 파일 입출력 개요

- 프로그램의 실행/종료 여부와 무관하게 데이터를 보존하고 싶다면??

→ 파일 입출력

✓ 데이터를 파일로부터 읽거나 파일로 출력하는 것

✓ C 언어는 파일 입출력 함수를 라이브러리 함수로 제공



[실습] 다음 프로그램을 실행시키고, 실제 파일이 생성되었는지 확인해보자

main.c

```
#include <stdio.h>

int main()
{
    double weight = 78.3;
    int age = 31;

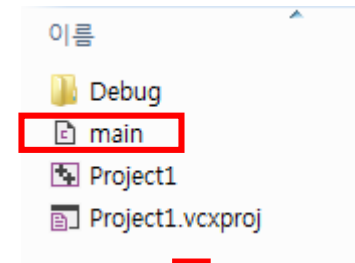
    FILE *fp;
    fp = fopen("test.txt", "w");

    fprintf(fp, "FIRST FILE TEST!\n");
    fprintf(fp, "%.2f %d\n", weight, age);

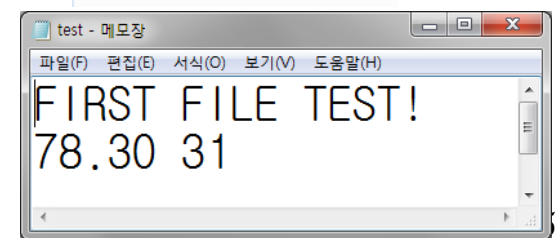
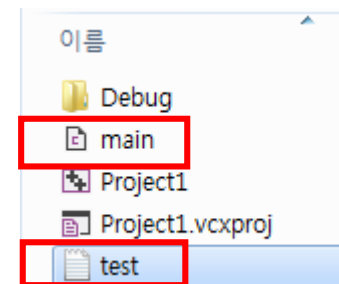
    fclose(fp);

    return 0;
}
```

현재 작업 중인 소스 프로그램 파일(main.c)이 있는 디렉터리



실행 결과: test.txt이 생성되고, 파일 내용은 다음과 같음!



[실습] 현재 작업 디렉터리에 "test_data.txt" 파일을 생성하고
다음 프로그램을 실행시켜보자.

```
#include <stdio.h>
#define SIZE 3
int main()
{
    double weight;
    int age, i;

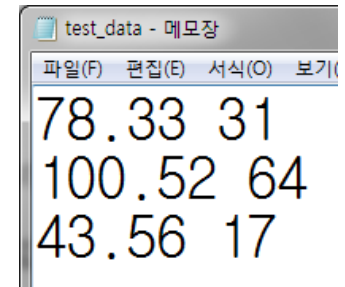
    FILE *fp;
    fp = fopen("test_data.txt", "r");

    for (i = 0; i < SIZE; i++)
    {
        fscanf(fp, "%lf %d", &weight, &age);
        printf("%.2f %d\n", weight, age);
    }

    fclose(fp);
    return 0;
}
```

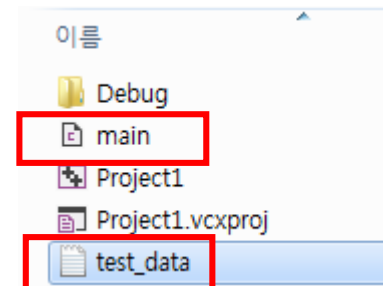
main.c

test_data.txt 파일 내용:
(메모장 사용해서 만들)

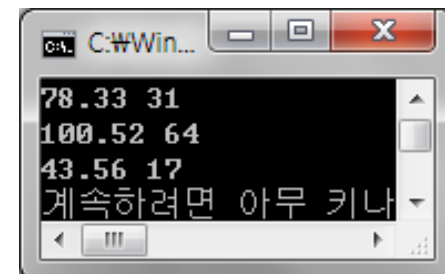


78.33 31
100.52 64
43.56 17

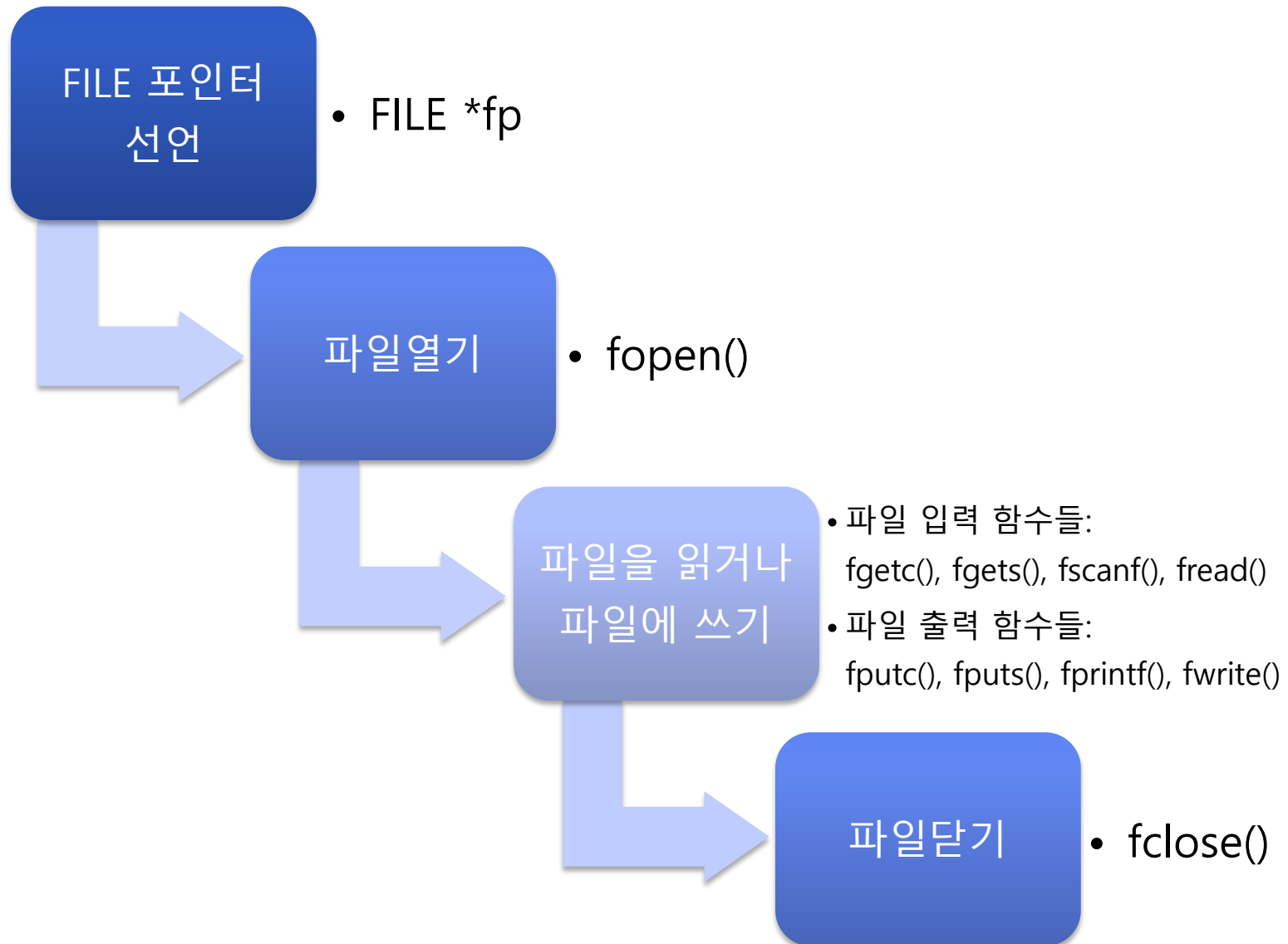
소스 프로그램 파일(main.c)이 있는 디렉터리에
test_data.txt 파일을 저장



실행결과



1) 파일 입출력 개요



목차

- 1) 파일 입출력 개요
- 2) 파일 입출력 절차**
- 3) 텍스트파일 vs. 이진파일
- 4) 텍스트파일의 입출력 함수
- 5) 이진파일의 입출력 함수 (심화 내용)
- 6) 기타 파일 입출력 관련 함수 (심화 내용)

2) 파일 입출력 절차

- 파일 입출력을 위해서는 **<stdio.h>** 파일을 반드시 포함해야 함
- 파일 포인터 (file pointer) 선언
 - 파일 포인터: FILE 구조체를 가리키는 포인터
 - FILE 구조체에 대한 포인터 변수를 선언하여 사용
 - 형식

FILE * 파일포인터명;



대문자!!

2) 파일 입출력 절차

▪ 파일열기: `fopen()` 함수

- 해당 파일에 대한 입출력 연결 형성을 요청하는 기능의 함수
- 해당 파일을 사용할 수 있도록 파일 포인터를 반환

함수원형	FILE *fopen(char *filename, char *filemode);	
함수인자	filename	입출력을 위해 연결 할 파일 이름
	filemode	연결할 파일 용도에 따른 모드
반환값	✓ 파일열기에 성공 → FILE 포인터를 반환 ✓ 파일열기에 실패 → NULL을 반환	

2) 파일 입출력 절차

- 함수인자 filename (1/3)

- fopen() 함수가 개방할 파일을 찾는 기본 위치

- ✓ 실행 방법, 실행환경 및 설정에 따라 개방할 파일이 있는 위치는 다를 수 있음

- ✓ 특정한 위치를 지정하지 않는 경우 → **현재 작업 디렉터리**

- ✓ 현재 작업 디렉터리 = 현재 작업하고 있는 소스 프로그램이 위치한 디렉터리

- ✓ 예) fopen("test.dat", "filemode");

큰 따옴표 내에, 해당 파일명만 작성해주면 됨!

2) 파일 입출력 절차

- 함수인자 filename (2/3)

- 개방할 파일이 현재 작업 디렉터리에 존재하지 않으면 → 경로를 함께 표기!

- ✓ 절대 경로 표기

- ✓ 드라이브 명과 디렉터리 경로를 포함하는 방식

- ✓ 실행하는 컴퓨터 환경과 상관없이 경로는 절대로 변경되지 않음

- ✓ 예) `fopen("C:₩₩C_pro₩₩Project₩₩test.dat", "filemode");`

- » C 드라이브의 하위 디렉터리인 C_pro, 이 디렉터리의 하위 디렉터리인 Project, 이 디렉터리 내에 test.dat 파일이 존재

- » ₩₩: ₩ 자체를 디렉터리를 나타내는 기호로 사용하기 위해서는 역슬래쉬(₩)를 두 번 사용해야 함

2) 파일 입출력 절차

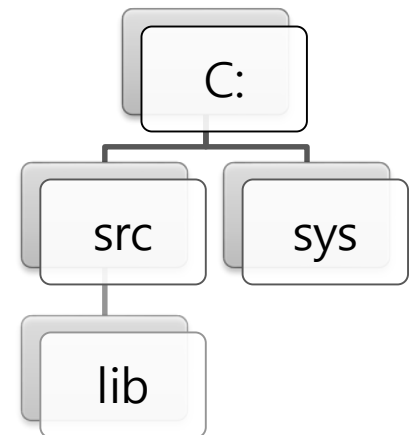
- 함수인자 filename (3/3)

- 상대 경로 표기

- ✓ 실행하는 컴퓨터 환경에 따라 경로가 바뀜
- ✓ 현재 작업 디렉터리 기준으로 상대경로를 지정
- ✓ 예) 그림 1에서 현재 작업 디렉터리: src

```
fopen("lib\\data.txt", "filemode");
```

```
fopen("../sys\\data2.txt", "filemode");
```



< 그림 1 >

2) 파일 입출력 절차

▪ 함수인자 filemode

- 개방할 파일의 용도에 따라 적합하게 지정해야 함
- 적합한 모드 지정은 파일을 잘못 사용하는 것을 막을 수 있음

〈 파일 접근 방식에 따른 모드 구분 〉

구분	모드	의미	기능
파일 입력	r	읽기 (read)	✓ 읽기 전용으로 오픈 ✓ 파일을 열 수 없는 경우 → NULL을 반환
파일 출력	w	쓰기 (write)	✓ 쓰기 전용으로 오픈 ✓ 파일이 없는 경우 → 새로 빈 파일을 생성 ✓ 같은 이름의 파일이 있는 경우 → 내용을 삭제하고 새로운 내용으로 파일을 생성
	a	추가 (append)	✓ 추가 쓰기 모드로 오픈 ✓ 파일이 없는 경우 → 새로 빈 파일을 생성 ✓ 같은 이름의 파일이 있는 경우 → 기존 파일의 마지막 부분에 내용을 추가

2) 파일 입출력 절차

- `fopen()` 함수 사용 예

```
FILE *fp;                //FILE 구조체 포인터  
fp = fopen("abc.txt", "w"); //abc.txt 파일을 쓰기 모드로 개방
```

```
FILE *fp2;  
fp2 = fopen("data/text.dat", "a");  
//현재 작업 디렉터리의 하위 디렉터리인 data 내 text.dat 파일을 추가 모드로  
개방
```

2) 파일 입출력 절차

- **fopen() 함수 사용 시 주의사항**
 - fopen() 함수 호출 시, 이 함수의 반환 값을 반드시 검사하여 파일이 정상적으로 열렸는지 확인해야 함

```
FILE *fp;  
fp = fopen("data.txt", "r");  
if (fp == NULL)  
{  
    printf("Couldn't open file!");  
    return -1;  
}
```


2) 파일 입출력 절차

- 파일닫기: `fclose()` 함수

- 해당 파일로의 입출력을 위한 연결을 닫음

함수 원형	int <code>fclose(FILE *fp);</code>	
함수 인자	fp	파일 포인터 변수명
반환 값	✓ 파일 닫기에 성공 → 0을 반환 ✓ 파일 닫기에 실패 → EOF를 반환	

※ **EOF (End Of File)**란?

- ✓ 파일의 끝을 표현하기 위해 정의해 놓은 상수 (즉, -1)
- ✓ 에러가 발생했는지 또는 파일 데이터를 모두 읽었는지 확인할 때 사용

2) 파일 입출력 절차

- `fclose()` 함수 사용 예

```
FILE *fp;  
fp = fopen("test.dat", "r");  
if (fp == NULL)  
{  
    printf("파일열기에 실패했습니다!\n");  
    return -1;  
}  
fclose(fp);  
printf("파일닫기에 성공했습니다!\n");
```

목차

- 1) 파일 입출력 개요
- 2) 파일 입출력 절차
- 3) **텍스트파일 vs. 이진파일**
- 4) 텍스트파일의 입출력 함수
- 5) 이진파일의 입출력 함수 (심화 내용)
- 6) 기타 파일 입출력 관련 함수 (심화 내용)

3) 텍스트파일 vs. 이진파일

- 파일 저장 방식에 따른 구분

	텍스트(text) 파일	이진(binary) 파일
특성	<ul style="list-style-type: none">✓사람이 인식할 수 있는 문자를 담고 있는 파일✓특별한 응용 프로그램 없이도 내용을 볼 수 있는 파일✓메모장 프로그램을 통해 파일을 열었을 때, 읽을 수 있는 문자들로 표현됨✓모든 데이터가 문자열로 변환되어 기록됨✓순차 처리 방식	<ul style="list-style-type: none">✓컴퓨터가 인식할 수 있는 데이터를 담고 있는 파일✓특정 응용 프로그램을 이용해야 액세스할 수 있는 파일✓메모장 프로그램을 통해 파일을 열었을 때, 알아볼 수 없는 이상한 문자들로 표현됨✓수치 데이터가 문자로 변환되지 않고 곧바로 수치로 저장✓텍스트 파일보다 저장 공간을 적게 차지✓읽고 쓰기가 빠름✓바이트 단위의 연속된 데이터 집합인 블록 단위로 데이터를 저장 → 임의 접근 처리 방식

3) 텍스트파일 vs. 이진파일

- **fopen() 함수인자 filemode (1/3)**

- ① 파일 접근 방식에 따른 모드 구분 (p. 14 참조)
- ② 파일 저장 방식에 따른 모드 구분

	텍스트(text) 모드	이진(binary) 모드
특성	<ul style="list-style-type: none">✓ 텍스트파일의 입출력 시에 사용✓ 운영체제마다 개행 표현 방식이 다른데, 이를 자동으로 변환해 줌✓ 개발자는 개행 문자의 변환을 신경 쓸 필요가 없음✓ 운영체제에 따른 표현 차로 인한 변환이 발생함	<ul style="list-style-type: none">✓ 이진파일의 입출력 시에 사용✓ 파일에 저장될 때도 이진 형식으로 표현된 내용이 그대로 파일에 저장✓ 행으로 분리되지 않으므로 행의 끝을 표시할 필요가 없음✓ 널 문자나 개행 문자 같은 글자들도 데이터로 취급✓ 숫자로만 이루어진 데이터를 파일에 저장할 경우 이진모드가 더 효율적임

3) 텍스트파일 vs. 이진파일

- `fopen()` 함수인자 `filemode` (2/3)

텍스트 모드	이진 모드	기능
r	rb	✓ 읽기 전용으로 오픈 ✓ 파일을 열 수 없는 경우 → NULL을 반환
w	wb	✓ 쓰기 전용으로 오픈 ✓ 파일이 없는 경우 → 새로 빈 파일을 생성 ✓ 같은 이름의 파일이 있는 경우 → 내용을 삭제하고 새로운 내용으로 파일을 생성
a	ab	✓ 추가 쓰기 모드로 오픈 ✓ 파일이 없는 경우 → 새로 빈 파일을 생성 ✓ 같은 이름의 파일이 있는 경우 → 기존 파일의 마지막 부분에 내용을 추가

3) 텍스트파일 vs. 이진파일

- `fopen()` 함수인자 `filemode` (3/3)

텍스트 모드	이진 모드	기능
r+	rb+	✓ 파일을 읽기와 쓰기 모드로 엮 ✓ 반드시 파일이 존재해야 함
w+	wb+	✓ 파일을 읽기와 쓰기 모드로 엮 ✓ 파일이 없는 경우 → 새로 빈 파일을 생성 ✓ 같은 이름의 파일이 있는 경우 → 내용을 삭제하고 새로운 내용으로 파일을 생성
a+	ab+	✓ 파일을 읽기와 추가 모드로 엮 ✓ 파일이 없는 경우 → 새로 빈 파일을 생성 ✓ 같은 이름의 파일이 있는 경우 → 읽기는 임의의 위치에서 가능하나, 쓰기는 파일의 끝에서만 가능

3) 텍스트파일 vs. 이진파일

- 텍스트모드를 사용한 `fopen()` 함수 사용 예

```
FILE *fp;                //FILE 구조체 포인터  
fp = fopen("test.txt", "r"); //test.txt 파일을 읽기모드로 개방
```

- 이진모드를 사용한 `fopen()` 함수 사용 예

```
FILE *fp;                //FILE 구조체 포인터  
fp = fopen("test.dat", "rb"); //test.dat 파일을 이진 읽기모드로 개방
```


3) 텍스트파일 vs. 이진파일

- 파일 입출력 함수

처리 대상	처리 단위	파일 입력	파일 출력
텍스트 파일	문자	fgetc()	fputc()
	문자열	fgets()	fputs()
	지정 형식	fscanf()	fprintf()
이진 파일	블록	fread()	fwrite()

※ C 언어에서는 위의 파일 입출력 함수를 라이브러리 함수(stdio.h)로 제공

3) 텍스트파일 vs. 이진파일

- 파일 입출력 함수

- 파일 입력 함수를 사용하기 위한 fopen() 함수인자 설정
 - ✓ 파일이름: 읽을 파일의 이름
 - ✓ 파일모드 (읽기모드)
 - ✓ 텍스트파일: "r"
 - ✓ 이진파일: "rb"
- 파일 출력 함수를 사용하기 위한 fopen() 함수인자 설정
 - ✓ 파일이름: 출력할 파일의 이름
 - ✓ 파일모드 (쓰기 혹은 추가모드)
 - ✓ 텍스트파일: "w" 혹은 "a"
 - ✓ 이진파일: "wb" 혹은 "ab"

목차

- 1) 파일 입출력 개요
- 2) 파일 입출력 절차
- 3) 텍스트파일 vs. 이진파일
- 4) 텍스트파일의 입출력 함수**
- 5) 이진파일의 입출력 함수 (심화 내용)
- 6) 기타 파일 입출력 관련 함수 (심화 내용)

4) 텍스트파일 입출력 함수

- 지정 형식 단위의 파일 입력 함수: **fscanf()**
 - 형식을 지정하여 파일의 데이터를 읽기 위한 함수
 - 여러 형태의 자료들(정수, 문자, 문자열 등)을 한번에 입력 가능
 - 함수의 첫 번째 인자로 파일 포인터가 사용된다는 것을 제외하고는 scanf() 함수와 사용법이 동일함

함수 원형	int fscanf(FILE *fp, char *format, arg1, arg2, ...);	
함수 인자	fp	파일 포인터 변수명
	format	형식 제어 문자열
	arg1, arg2, ...	입력하고자 하는 변수리스트
반환 값	✓ 성공 → 입력한 변수의 개수를 반환 ✓ 파일의 끝이거나 오류 발생 → EOF를 반환	

4) 텍스트파일 입출력 함수

- **fscanf() 함수 사용 예**

```
char str[10];  
int num;  
FILE *fp = fopen("data.txt", "r");  
if (fp == NULL)  
{  
    printf("Couldn't open file!");  
    return -1;  
}  
  
fscanf(fp, "%s %d", str, &num);
```

CHOI 12
LEE 13

➔ fp에 연결된 data.txt 파일로부터 문자열과 정수를 읽어와서 각각 str 배열과 num에 저장

4) 텍스트파일 입출력 함수

- 지정 형식 단위의 파일 출력 함수: **fprintf()**
 - 형식을 지정하여 파일에 데이터를 쓰기 위한 함수
 - 함수의 첫 번째 인자로 파일 포인터가 사용된다는 것을 제외하고는 **printf()** 함수와 사용법이 동일함

함수 원형	int fprintf(FILE *fp, char *format, arg1, arg2, ...);	
함수 인자	fp	파일 포인터 변수명
	format	형식 제어 문자열
	arg1, arg2, ...	출력하고자 하는 변수리스트
반환 값	✓ 성공 → 출력한 데이터의 바이트 수 ✓ 실패/오류 발생 → 음수를 반환	

4) 텍스트파일 입출력 함수

▪ fprintf() 함수 사용 예

```
int age = 25;
FILE *fp = fopen("data.txt", "w");
if (fp == NULL)
{
    printf("Couldn't open file!");
    return -1;
}
```

fprintf(fp, "나이: %d세", age);

fprintf(stdout, "나이: %d세", age); // printf("나이: %d세", age);

➔ fp에 연결된 data.txt 파일과 모니터에 동일하게 "나이: 25세"가 출력됨

FILE 포인터 이름	스트림	의미
stdin	표준 입력 스트림	키보드로부터 입력 받음
stdout	표준 출력 스트림	모니터로 결과 출력
stderr	표준 오류 출력 스트림	모니터로 오류 메시지 출력



4) 텍스트파일 입출력 함수

▪ 문자 단위의 파일 입력 함수: fgetc()

- 문자 한 개를 파일로부터 읽기 위한 함수

함수 원형	int fgetc(FILE *fp);	
함수 인자	fp	파일 포인터 변수명
반환 값 (char형 이 아니고 int형)	✓ 성공 → 파일로부터 읽은 문자를 반환 ✓ <u>파일의 끝</u> 에 도달하거나 <u>오류발생</u> → EOF를 반환	

▪ 문자 단위의 파일 출력 함수: fputc()

- 문자 하나를 파일에 쓰기 위한 함수

함수 원형	int fputc(int char, FILE *fp);	
함수 인자	char	출력하고자 하는 문자 상수 또는 변수
	fp	파일 포인터 변수명
반환 값	✓ 성공 → 출력하는 문자 char를 반환 ✓ 실패/오류발생 → EOF를 반환	

4) 텍스트파일 입출력 함수

▪ [실습] 문자 단위의 파일 입출력 함수 연습

```
#include <stdio.h>

int main()
{
    FILE *fp1, *fp2;
    char ch;

    fp1 = fopen("input.txt", "r");
    if (fp1 == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
    fp2 = fopen("output.txt", "w");
    if (fp2 == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
}
```

```
while((ch = fgetc(fp1)) != EOF)
{
    printf("%c", ch);
    fputc(ch, fp2);
}

fclose(fp1);
fclose(fp2);

return 0;
}
```

4) 텍스트파일 입출력 함수

- 문자열 단위의 파일 입력 함수: `fgets()`
 - 파일에 쓰여진 문자열을 읽는데 사용하는 함수
 - 파일에 쓰여진 **개행 문자까지 문자열에 포함**
 - 한 번에 읽을 수 있는 문자열의 길이가 정해져 있음
 - ✓ 한 번에 읽을 수 있는 문자열
= (최대 입력할 수 있는 문자 수 - 1) 개의 문자 + 널 문자
 - 한 번에 읽을 수 있는 문자열 내에, 개행 문자가 포함되어 있다면
➔ 개행 문자까지의 문자열을 반환함

4) 텍스트파일 입출력 함수

- 문자열 단위의 파일 입력 함수: `fgets()`

함수 원형	<code>char *fgets(char *s, int n, FILE *fp);</code>	
함수 인자	s	파일로부터 읽을 문자열을 저장할 포인터
	n	읽을 문자열의 최대 길이
	fp	파일 포인터 변수명
반환 값	✓ 성공 → 문자열 s를 반환 ✓ 파일의 끝에 도달하거나 실패/오류 발생 → NULL을 반환	

4) 텍스트파일 입출력 함수

▪ fgets() 함수 사용 예

```
char str1[20], str2[20], str3[20];  
FILE *fp = fopen("info.txt", "r");
```

info.txt

Neungdong-ro, ↵
Gwangjin-gu, Seoul, Korea. ↵

1) fgets(str1, 20, fp);

N	e	u	n	g	d	o	n	g	-	r	o	,	↵	↵					
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	--	--	--	--	--

→ 20개의 문자를 읽기 전에 개행 문자를 읽으므로 뒤에 널 문자를 합쳐 str1 배열에 저장

2) fgets(str2, 20, fp);

G	w	a	n	g	j	i	n	-	g	u	,		S	e	o	u	l	,	↵
---	---	---	---	---	---	---	---	---	---	---	---	--	---	---	---	---	---	---	---

→ 19개의 문자를 읽은 후 뒤에 널 문자를 합쳐 str2 배열에 저장

3) fgets(str3, 20, fp);

	K	o	r	e	a	.	↵	↵											
--	---	---	---	---	---	---	---	---	--	--	--	--	--	--	--	--	--	--	--

→ 이전에 읽은 곳 다음부터 읽다가 20개의 문자를 읽기 전에 개행 문자를 읽게 되어, 그 뒤에 널 문자를 합쳐 str3 배열에 저장

4) 텍스트파일 입출력 함수

- 문자열 단위의 파일 출력 함수: `fputs()`
 - 문자열을 파일에 쓰기 위한 함수
 - 문자열의 끝을 나타내는 널 문자는 파일에 쓰지 않으며, 그 뒤에 개행 문자도 자동으로 들어가지 않음

함수 원형	<code>int fputs(char *str, FILE *fp);</code>	
함수 인자	str	출력하고자 하는 문자열 상수 또는 변수
	fp	파일 포인터 변수명
반환 값	✓ 성공 → 출력한 바이트 수를 반환 ✓ 실패/오류발생 → EOF를 반환	

(비교) `puts()` 함수가 호출되면 문자열 출력 후 **자동으로** 개행이 이루어진다

4) 텍스트파일 입출력 함수

▪ [실습] 문자열 단위의 파일 입출력 함수 연습

```
#include <stdio.h>

int main()
{
    char str[100];
    FILE *fp1, *fp2;

    fp1 = fopen("input.txt", "r");
    if (fp1 == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
    fp2 = fopen("output.txt", "w");
    if (fp2 == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
}
```

```
while(fgets(str, sizeof(str), fp1) != NULL)
{
    printf("%s", str);
    fputs(str, fp2);
}

fclose(fp1);
fclose(fp2);

return 0;
}
```

4) 텍스트파일 입출력 함수

- 파일의 끝을 확인하는 함수: feof()

- 파일의 끝까지 데이터를 모두 읽어 들인 상태인지를 확인하는 함수
- 이 함수를 사용하기 위해서는 <stdio.h>를 포함시켜야 함

함수 원형	int feof(FILE *fp);	
함수 인자	fp	파일 포인터 변수명
반환 값	✓ 파일의 끝이면 → 0이 아닌 값을 반환 ✓ 파일의 끝이 아니면 → 0을 반환	

- EOF vs. feof() 함수

- 모든 파일의 끝에는 EOF가 존재 → EOF는 파일의 한 부분임
- feof() 함수는 EOF를 만났을 때, 0을 반환!
- 그 이후에 0이 아닌 값을 반환!

4) 텍스트파일 입출력 함수

▪ feof() 함수 사용 시, 주의사항

- (예제 1) data.txt 파일을 빈 파일로 하여, 다음 프로그램을 실행시켜 보자. 결과는?

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

```
int main()  
{
```

```
    FILE *fp;  
    char str[100];
```

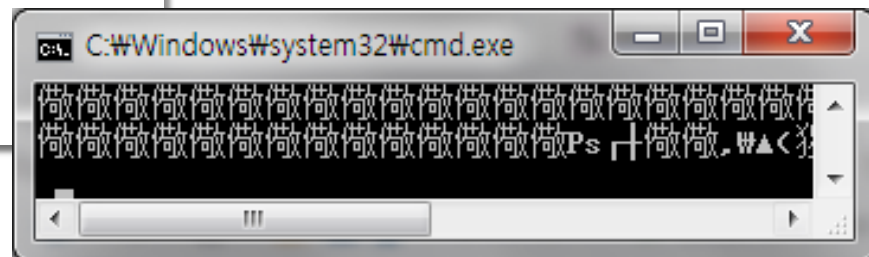
```
    fp = fopen("data.txt", "r");
```

```
    if (fp == NULL)  
    {
```

```
        printf("Couldn't open file!");  
        return -1;
```

```
    }
```

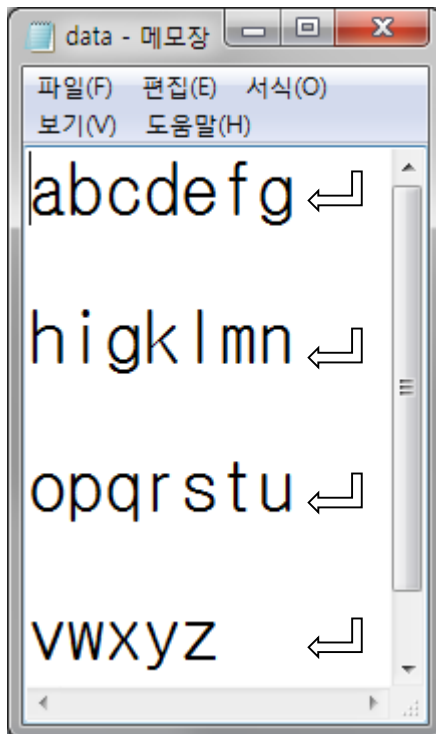
```
while(!feof(fp))  
{  
    fgets(str, sizeof(str), fp);  
    printf("%s", str);  
}  
fclose(fp);  
return 0;  
}
```



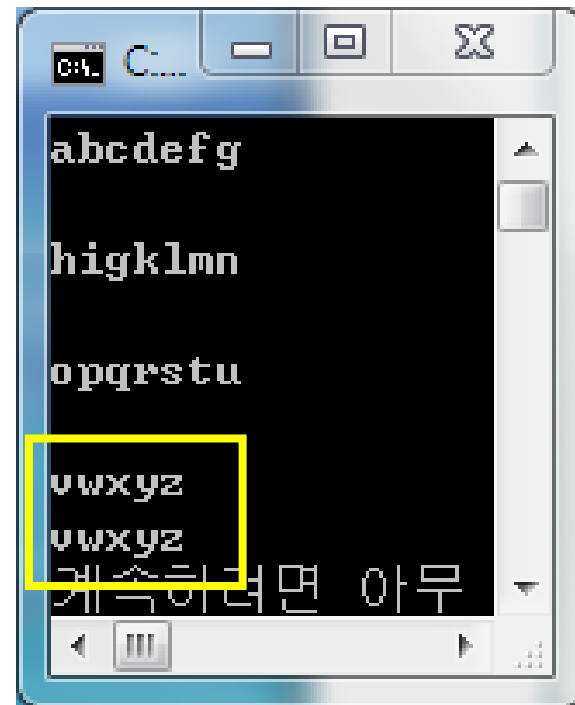
4) 텍스트파일 입출력 함수

- **feof() 함수 사용 시, 주의사항**
 - (예제 2) 다음과 같은 내용의 data.txt 파일을 생성하고, 앞의 프로그램을 실행시켜 보자. 결과는?

<data.txt 파일 내용>



<실행 결과>



4) 텍스트파일 입출력 함수

- **feof() 함수 사용 시, 주의사항**

- 앞의 예들의 원인

- ✓ 마지막 데이터 뒤에는 파일의 끝을 나타내는 특수 문자(^Z)가 시스템에 의해 자동으로 들어감

- ✓ 빈 파일의 경우에도 ^Z가 존재함(예1)

- ✓ ^Z의 위치 문제(예2)

- ✓ feof(fp) 함수 반환 값 = 0(거짓)

- ✓ ^Z를 지나가야 feof(fp) 함수는 0이 아닌 값(참)을 반환함

- 해결방법

- ✓ 먼저 데이터를 읽은 후, 파일의 끝에 도달했는지를 확인하도록 함

4) 텍스트파일 입출력 함수

- **feof() 함수 사용 시, 주의사항**
 - 앞의 문제들을 해결한 수정된 코드

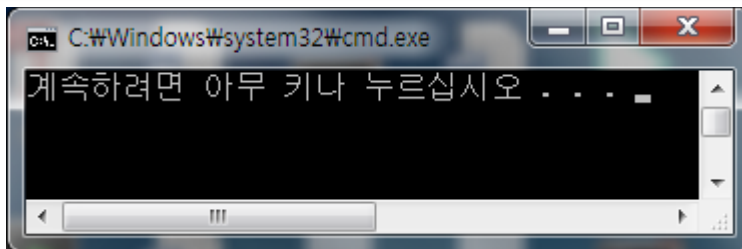
```
#include <stdio.h>

int main()
{
    FILE *fp;
    char str[100];

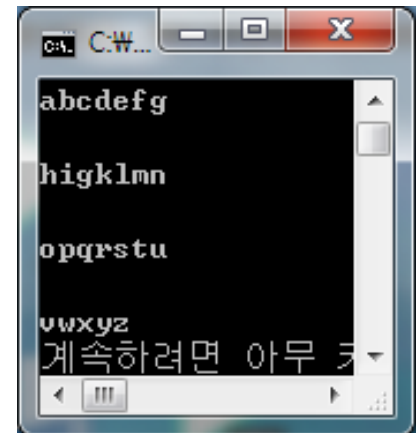
    fp = fopen("data.txt", "r");
    if (fp == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
```

```
fgets(str, sizeof(str), fp);
while(!feof(fp))
{
    printf("%s", str);
    fgets(str, sizeof(str), fp);
}
fclose(fp);
return 0;
}
```

<예제1
실행 결과>



<예제2
실행 결과>



The END

목차

- 1) 파일 입출력 개요
- 2) 파일 입출력 절차
- 3) 텍스트파일 vs. 이진파일
- 4) 텍스트파일의 입출력 함수
- 5) 이진파일의 입출력 함수 (심화 내용)
- 6) 기타 파일 입출력 관련 함수 (심화 내용)

5) 이진파일 입출력 함수 (심화 내용)

- 블록 단위의 파일 입출력 함수
 - 이진파일을 대상으로 함
 - 한 번에 지정한 크기의 블록 단위로 데이터를 파일로부터 읽거나 쓰고자 할 때 사용
 - ✓ 블록(block): 바이트 단위의 연속된 데이터 집합
 - 구조체와 같이 일정한 크기의 데이터를 입출력해야 할 때, 주로 사용됨
 - fread() 함수와 fwrite() 함수가 있음

5) 이진파일 입출력 함수 (심화 내용)

- 이진파일 입력 함수: `fread()`
 - 이진파일에서 데이터 블록을 읽기 위한 함수

함수 원형	unsigned int fread(void *ptr, unsigned int size, unsigned int n, FILE *fp);		
함수 인자	ptr	파일로부터 읽은 데이터를 기억시킬 버퍼의 시작 주소	
	size	읽을 데이터의 바이트 수 (블록의 크기)	
	n	size만큼 읽기 위한 반복 횟수 (블록의 개수)	
	fp	파일 포인터 변수명	
반환 값	✓ 성공 → 파일로부터 읽은 블록의 개수 n을 반환 ✓ 파일의 끝 혹은 실패 → n보다 작은 값을 반환		
의미	이진파일에서 (size * n) 바이트의 데이터를 읽어 버퍼에 저장한 후, 읽은 블록 개수를 반환		

5) 이진파일 입출력 함수 (심화 내용)

- **fread() 함수 사용 예**

```
int height, age[10];
```

```
FILE *fp = fopen("data.bin", "rb");
```

```
fread(&height, sizeof(int), 1, fp);
```

➔ fp에 연결된 이진파일에서 int형 크기의 블록 1개를 읽어서 변수 height의 메모리에 씀

➔ 즉, 파일에서 1개의 정수를 읽어서 height 변수에 저장

```
fread(age, sizeof(int), 10, fp);
```

➔ fp에 연결된 이진파일에서 int형 크기의 블록 10개를 읽어서 age (배열의 시작주소) 번지의 메모리에 저장

➔ 즉, 파일에서 10개의 정수를 읽어서 age 배열에 저장

5) 이진파일 입출력 함수 (심화 내용)

- 이진파일 출력 함수: `fwrite()`
 - 이진파일에 데이터 블록을 쓰기 위한 함수

함수 원형	unsigned int fwrite(const void *ptr, unsigned int size, unsigned int n, FILE *fp);		
함수 인자	ptr	파일에 기록하려는 데이터가 있는 버퍼의 시작 주소	
	size	파일에 출력하는 데이터의 바이트 수 (블록의 크기)	
	n	size만큼 쓰기 위한 반복 횟수 (블록의 개수)	
	fp	파일 포인터 변수명	
반환 값	✓ 성공 → 파일에 출력한 블록의 개수 n을 반환 ✓ 실패 → n 보다 작은 값을 반환		
의미	버퍼에 저장된 (size * n) 바이트의 데이터를 이진파일에 출력한 후, 출력한 블록 개수를 반환		

5) 이진파일 입출력 함수 (심화 내용)

- **fwrite() 함수 사용 예**

```
Int height, age[10];
```

```
FILE *fp = fopen("data.bin", "wb");
```

```
fwrite(&height, sizeof(int), 1, fp);
```

➔ 변수 height 메모리에서 int형 크기의 블록 1개를 읽어서 fp에 연결된 이진파일에
쓰

➔ 즉, height 변수에서 1개의 정수를 읽어서 data.bin 파일에 저장

```
fwrite(age, sizeof(int), 10, stdout);
```

➔ age 배열에서 int형 크기의 블록 10개를 읽어서 표준출력장치인 모니터로 출력

➔ 즉, age 배열에서 10개의 정수를 읽어 모니터에 출력

5) 이진파일 입출력 함수 (심화 내용)

[실습] 이진파일 입출력 함수 연습

```
#include <stdio.h>
struct person{
    char name[8];
    int age;
} data[10]={{"Tom",46}, {"James",33}, {"Jane",21}};

void main()
{
    FILE *fp;
    struct person buf[10];
    int i;

    fp=fopen("data.txt", "w");
    fwrite(data, sizeof(struct person), 3, fp);
    fclose(fp);

    fp=fopen("data.txt", "r");
    fread(buf, sizeof(struct person), 3, fp);
    for(i=0; i<3; i++){
        printf("i=%d %s %d\n", i, buf[i].name, buf[i].age);
    }
    fclose(fp);
}
```

목차

- 1) 파일 입출력 개요
- 2) 파일 입출력 절차
- 3) 텍스트파일 vs. 이진파일
- 4) 텍스트파일의 입출력 함수
- 5) 이진파일의 입출력 함수 (심화 내용)
- 6) 기타 파일 입출력 관련 함수 (심화 내용)

6) 기타 파일 입출력 관련 함수 (심화 내용)

- 파일의 임의 접근 처리 방식

- 이진파일을 대상으로 함
- 파일의 임의의 위치에서 바로 읽기/쓰기를 할 수 있는 접근 방식
- 파일 읽기/쓰기를 시작할 위치를 가리키는 포인터인 파일 위치 지시자를 조작하는 함수를 사용
 - ✓ 파일 위치 지시자: 파일에서 다음에 읽거나 쓸 데이터의 위치를 나타냄
- `fseek()`, `rewind()`, `ftell()` 함수 사용 가능
 - ✓ 이 함수들을 이용하면 이진파일에 대해 임의의 블록을 곧바로 찾아가서 읽거나 블록을 수정하는 작업이 가능
 - ✓ 이 함수들을 사용하기 위해서는 `<stdio.h>` 파일을 포함시켜야 함



6) 기타 파일 입출력 관련 함수 (심화 내용)

▪ fseek() 함수 (1/2)

- 파일 위치 지시자를 지정한 위치로 이동시킬 수 있는 함수
- fp에 연결된 파일의 파일 위치 지시자가 origin으로부터 offset만큼 떨어진 곳을 가리키게 함
- 다음에 읽기/쓰기를 시작할 위치를 (origin + offset) 바이트 위치로 변경함
- origin 값: 상수로 정의된 SEEK_SET(0), SEEK_CUR(1), SEEK_END(2) 중 하나를 사용

6) 기타 파일 입출력 관련 함수 (심화 내용)

▪ fseek() 함수 (2/2)

함수 원형	int fseek(FILE *fp, long int offset, int origin);	
함수 인자	fp	파일 포인터 변수명
	offset	✓ origin으로부터 이동할 바이트 수 - 양수(+): 순방향(기준점 이후) - 음수(-): 역방향(기준점 이전)
	origin	✓ offset을 적용할 기준점 - SEEK_SET(0): 파일의 맨 처음 위치 - SEEK_CUR(1): 파일에서의 현재 위치 - SEEK_END(2): 파일의 맨 끝 위치
반환 값	✓ 성공 → 0을 반환 ✓ 실패 → 0이 아닌 값을 반환	

6) 기타 파일 입출력 관련 함수 (심화 내용)

▪ fseek() 함수 사용 예

- `fseek(fp, 10, SEEK_SET); // 0`

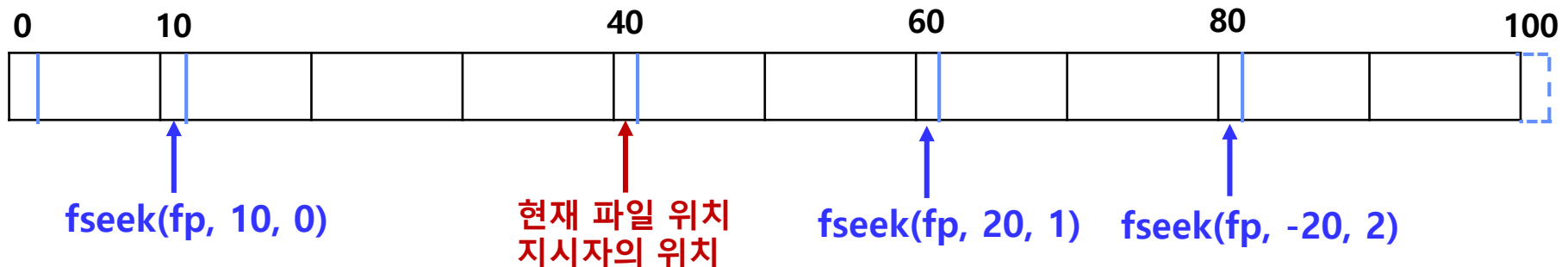
→ 다음 읽기/쓰기 위치를 파일 시작 지점에서 10바이트 이후로 이동

- `fseek(fp, 20, SEEK_CUR); // 1`

→ 다음 읽기/쓰기 위치를 현재 위치에서 20바이트 이후로 이동

- `fseek(fp, -20, SEEK_END); // 2`

→ 다음 읽기/쓰기 위치를 파일의 끝 지점에서 20바이트 이전으로 이동



6) 기타 파일 입출력 관련 함수 (심화 내용)

▪ rewind() 함수

- 파일 위치 지시자를 파일의 시작 지점으로 이동시키는 함수
- fseek(fp, 0, SEEK_SET)와 동일한 효과

함수 원형	void rewind(FILE *fp)	
함수 인자	fp	파일 포인터 변수명

6) 기타 파일 입출력 관련 함수 (심화 내용)

▪ ftell() 함수

- 현재 파일 위치 지시자가 가리키는 곳의 위치를 반환하는 함수
- 즉, 현재 파일 위치 지시자가 가리키는 곳이 파일의 시작 위치로부터 몇 바이트 떨어져 있는지를 알려줌
 - ✓ 가정사항) 파일의 시작 위치를 상대적 위치 0으로 간주함!

함수 원형	long ftell (FILE *fp);	
함수 인자	fp	파일 포인터 변수명
반환 값	✓ 성공 → 읽기/쓰기 위치를 반환 ✓ 실패/오류 → -1을 반환	

6) 기타 파일 입출력 관련 함수 (심화 내용)

- [실습] fseek() 함수와 ftell() 함수 사용 예

```
#include <stdio.h>
int main()
{
    FILE *fp;
    int size;
    fp = fopen("data.txt", "rb");
    if (fp == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    fclose(fp);
    printf("Size of the file: %d bytes.\n", size);
    return 0;
}
```

6) 기타 파일 입출력 관련 함수 (심화 내용)

- 일반적으로, 파일 버퍼 내의 데이터가 파일에 출력되는 시점
 - ① 버퍼가 다 찼을 때
 - ② 파일이 닫힐 때
 - ③ 프로그램이 종료될 때
- 버퍼 플러쉬(buffer flush)란?
 - 필요에 의해 버퍼의 내용을 강제로 파일에 출력하고, 버퍼를 비우는 일을 말함
 - fflush() 함수 사용
 - ✓ 이 함수를 사용하기 위해서는 <stdio.h> 파일을 포함시켜야 함

함수 원형	int fflush(FILE *fp);	
함수 인자	fp	파일 포인터 변수명
예	fflush(stdout) → 출력 버퍼 안에 존재하는 데이터들은 즉시 출력됨	

6) 기타 파일 입출력 관련 함수 (심화 내용)

[실습]

fflush() 함수 사용 예

```
#include <stdio.h>
char mybuf[30];
int main()
{
    FILE *fp;
    fp = fopen("data.txt", "r+");
    if (fp == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
    fputs("Remove data (fflush) ", fp);
    fflush(fp);
    fgets(mybuf, 30, fp);
    puts(mybuf);
    fclose(fp);
    return 0;
}
```

-
- "C언어에 포커스를 두기 위해서"
 - p.5 스트림 설명, 페이지 삭제
 - p.18 fclose 함수 : 운영체제에서 버퍼를 비운다 (설명 삭제)
 - 전반적으로
 - 스트림 ---> 연결로 수정
 - p.25 파일모드에서 텍스트 파일 읽기 모드 "r(rt)" ---> "r" 로 수정합니다.
rt 는 표준이 아닙니다.
 - p. 43 파일의 끝 확인하는 두 가지 방법 삭제
(파일 끝 확인 가능하지만, 파일 끝이 아니어도 동일 값을 반환한다)
feof() 함수만 남겨두었습니다.