

# C 프로그래밍 1

## Lecture Note #19

---

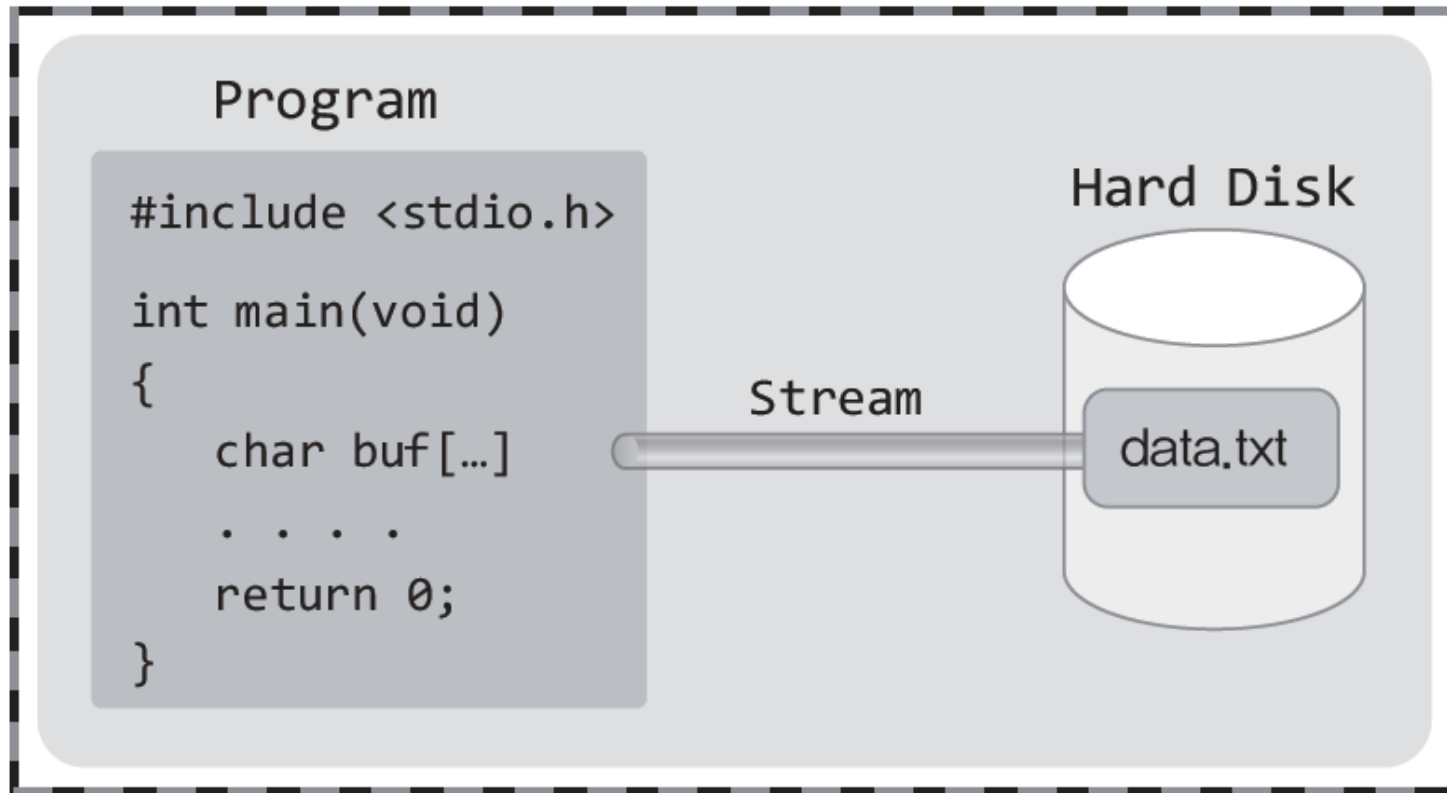
백윤철  
ybaek@smu.ac.kr

# 내용

---

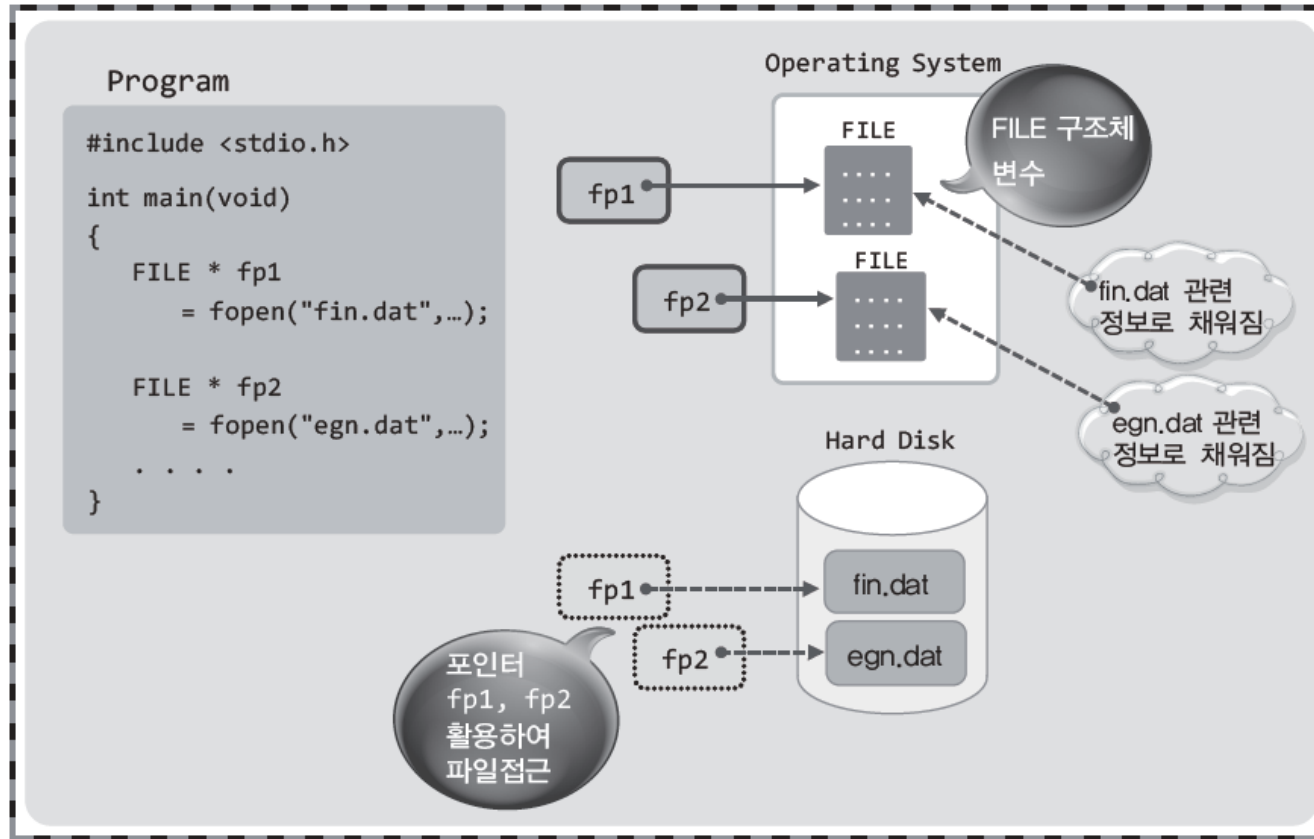
- 기본 입출력
- file open mode
- 파일 입출력 함수
- 서식이 있는 입출력
- 파일 읽기/쓰기 위치 변경

# 파일에 저장되어 있는 데이터를 읽고 싶어요



- ❑ 콘솔 입출력과 마찬가지로 파일로부터의 데이터 입출력을 위해서는 **스트림**이 형성되어야 함
- ❑ 파일과의 스트림 형성은 데이터 입출력의 기본

# fopen 함수를 통한 스트림의 형성



- fopen 함수 호출시 생성되는 **FILE 구조체 변수**와 이를 참조하는 **FILE 구조체 포인터 변수**의 관계

# fopen 함수 호출의 결과

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

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

→ 성공 시 해당 파일의 FILE 구조체 변수의 주소 값, 실패 시 NULL 포인터 반환

- fopen 함수가 호출되면 FILE 구조체 변수가 생성됨
  - 생성된 FILE 구조체 변수에는 파일에 대한 정보가 담기게 됨
  - FILE 구조체의 포인터는 파일을 가리키는 '지시자'에 대응함
- fopen 함수가 파일과의 스트림 형성을 요청하는 기능의 함수이다.

# 출력 스트림의 생성

"w"에는 출력  
스트림의 의미가  
담겨있다.

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

"파일 data.txt"와 스트림을 형성하되 ~~w~~ 모드로 스트림  
을 형성할 것



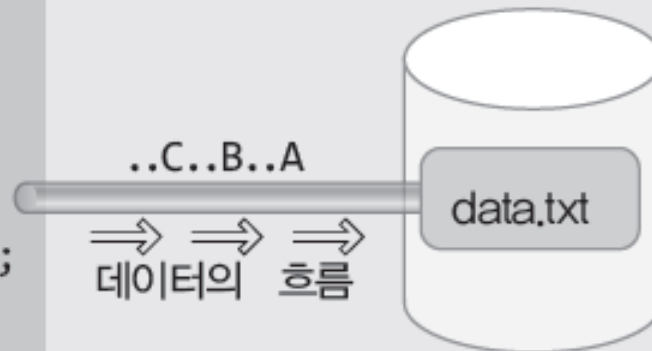
출력 스트림 형성 결과

Program

```
#include <stdio.h>

int main(void)
{
    FILE * fp=fopen(...);
    . . . .
}
```

Hard Disk



포인터 변수  
fp에 저장된  
값이  
data.txt의  
스트림에  
데이터를  
전송하는  
도구가 된다.

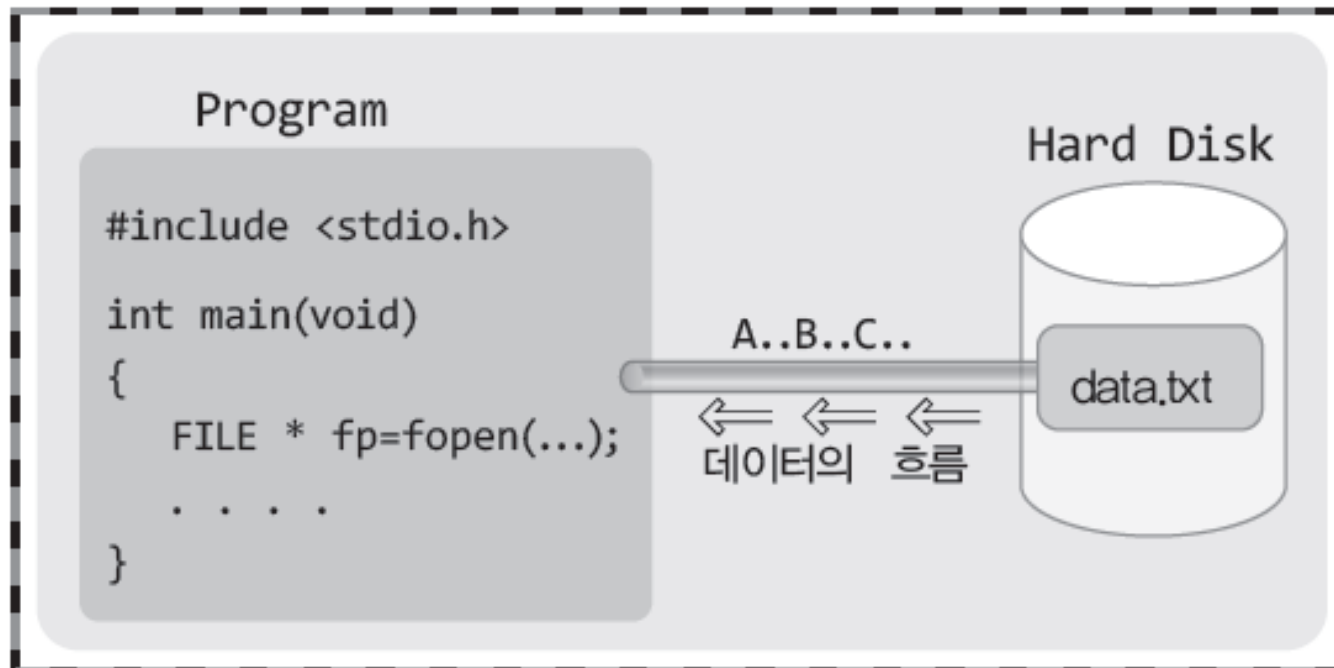
# 입력 스트림의 생성

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

"파일 data.txt"와 스트림을 형성하되 r 모드로 스트림을 형성할 것



입력 스트림 형성 결과



포인터 변수  
fp에 저장된  
값이  
data.txt의  
스트림으로부  
터 데이터를  
수신하는  
도구가 된다.

# 파일에 데이터를 써봅시다

```
int main(void) {  
    FILE *fp = fopen("data.txt", "w");  
    if (fp == NULL) {  
        puts("파일 오픈 실패!");  
        return -1;  
    }  
    fputc('A', fp);  
    fputc('B', fp);  
    fputc('C', fp);  
    fclose(fp);  
    return 0;  
}
```

현재 디렉터리에 저장된 파일  
data.txt를 찾는다.

현재 디렉터리는 실행파일이  
저장된 디렉터리이거나  
프로젝트 파일이 저장된  
디렉터리이다!

FILE \* fp = fopen("C:\\Project\\data.txt", "w");  
**fopen 함수호출 시 경로를 완전히 명시 가능**



# 스트림의 소멸을 요청하는 fclose 함수

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

```
int fclose(FILE *stream);
```

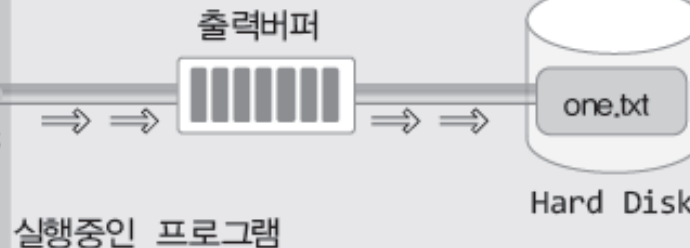
→ 성공 시 0, 실패 시 EOF를 반환

fclose 함수호출이  
동반하는 두 가지

- 운영체제가 할당한  
자원의 반환
- 버퍼링 되었던  
데이터의 출력

```
int main(void)
```

```
{  
    FILE * fp1;  
    fp1=fopen("one.txt", "wt" );  
    . . . . .  
}
```



```
int main(void)
```

```
{  
    FILE * fp2;  
    fp1=fopen("two.txt", "rt" );  
    . . . . .  
}
```



fclose 함수가 호출되어야 스트림 형성 시 할당된 모든 리소스가 소멸이 된다. 따라서 파일이 오픈 된 상태로 놔두는 것은 좋지 않다.

# fflush 함수

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

```
int fflush(FILE *stream);
```

→ 성공 시 0, 실패 시 EOF를 반환

- 출력버퍼를 비운다는 것은 출력버퍼에 저장된 데이터를 목적지로 전송한다는 의미
- 입력버퍼를 비운다는 것은 입력버퍼에 저장된 데이터를 소멸시킨다는 의미
- fflush 함수는 출력버퍼를 비우는 함수
- fflush 함수는 입력버퍼를 대상으로 호출할 수 없음

# fflush 함수

```
#include <stdio.h>
int main(void)
{
    FILE *fp = fopen("data.txt", "w");
    ...
    fflush(fp); /* 출력 버퍼 비우기 요청 */
    ...
}
```


이렇듯 fflush 함수의 호출을 통하여 fclose 함수를 호출하지 않고도 출력버퍼만 비울 수 있다.

그렇다면 파일의 입력버퍼는 어떻게 비우는가?

# 파일로부터 데이터를 읽어봅시다

```
#include <stdio.h>
int main(void) {
    FILE *fp = fopen("data.txt", "r");
    if (fp == NULL) {
        puts("파일 오픈 실패!");
        return -1;
    }
    for (i = 0; i < 3; i++) {
        ch = fgetc(fp);
        printf("%c\n", ch);
    }
    fclose(fp); return 0; }
```

fp로부터 하나의  
문자를 읽어서  
변수 ch에  
저장해라!



# 스트림의 구분 기준 두 가지 (Basic)

## □ 기준 1

- 읽기 위한 스트림? 쓰기 위한 스트림?

## □ 기준 2

- 텍스트 데이터를 위한 스트림? 바이너리 데이터를 위한 스트림?



기본적인  
스트림의 구분  
그러나  
실제로는 더  
세분화

# 스트림을 구분하는 기준 1: Read or Write

- 스트림의 성격은 R/W를 기준으로 다음과 같이 세분화됨

| 모드(mode) | 스트림의 성격          | 파일이 없으면? |
|----------|------------------|----------|
| r        | 읽기 가능            | 에러       |
| w        | 쓰기 가능            | 생성       |
| a        | 파일의 끝에 덧붙여 쓰기 가능 | 생성       |
| r+       | 읽기/쓰기 가능         | 에러       |
| w+       | 읽기/쓰기 가능         | 생성       |
| a+       | 읽기/덧붙여 쓰기 가능     | 생성       |

- 모드의 +는 읽기/쓰기 모두 가능
- 모드의 a는 덧붙여 쓰기 가능

## 스트림을 구분하는 기준 2: 텍스트/바이너리

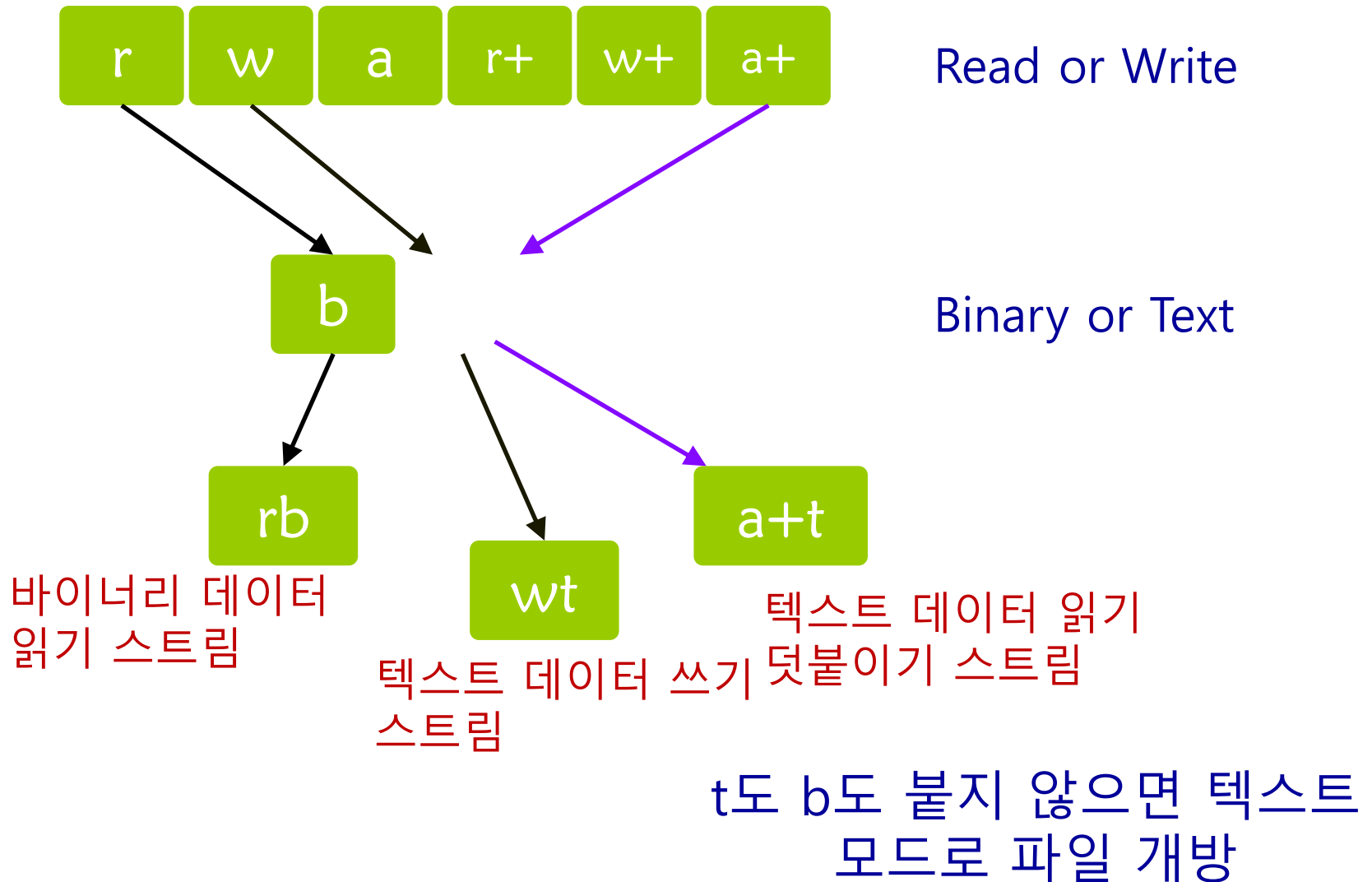
- 스트림의 성격은 데이터의 종류에 따라 다음과 같이 두 가지로 나누어짐
- 텍스트 모드 스트림 (t): 문자 데이터를 저장하는 스트림 *.hwp*
- 바이너리 모드 스트림 (b): 바이너리 데이터를 저장하는 스트림 *.jpg*

## 스트림을 구분하는 기준 2: 텍스트/바이너리

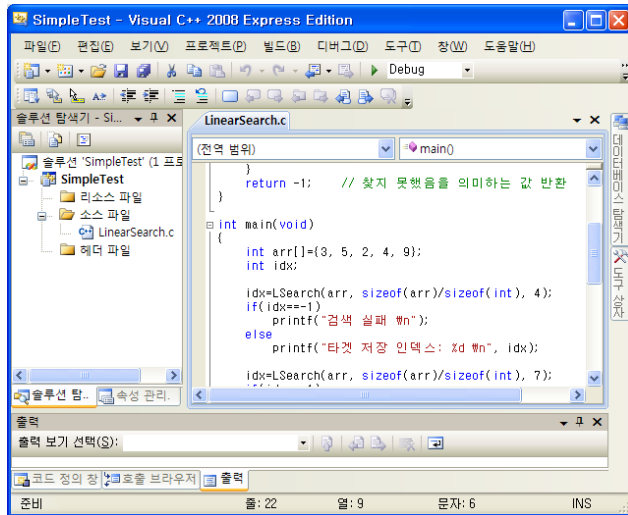
- 문자 데이터: 사람이 인식할 수 있는 유형의 문자로 이뤄진 데이터
  - 파일에 저장된 문자 데이터는 Windows의 메모장으로 열어 문자 확인
  - 예: 도서목록, 물품가격, 전화번호, 주민등록번호
  
- 바이너리 데이터: 컴퓨터가 인식할 수 있는 유형의 데이터
  - 메모장과 같은 편집기로는 그 내용이 의미하는 바를 알 수 없음
  - 예: 음원 및 영상 파일, 그래픽 디자인 프로그램에 의해 저장된 디자인 파일



# 파일의 개방모드 조합



# 텍스트 스트림이 별도로 존재하는 이유



C언어는 개행을 `\n`으로 표시하기로 약속하였다. 따라서 개행 정보를 저장할 때 C 프로그램상에서 우리는 `\n`을 저장한다..

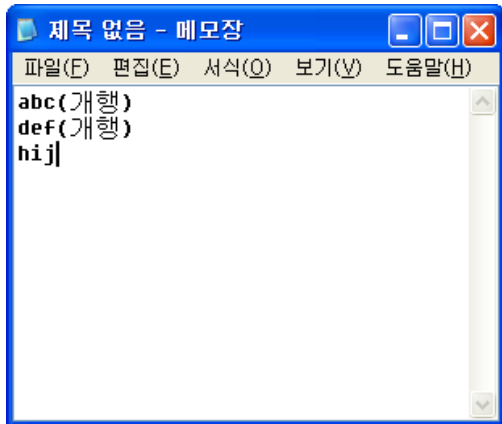


개행 정보로 저장된 `\n`은 문제가 되지 않을까?

text.txt

# 텍스트 스트림이 별도로 존재하는 이유

text.txt



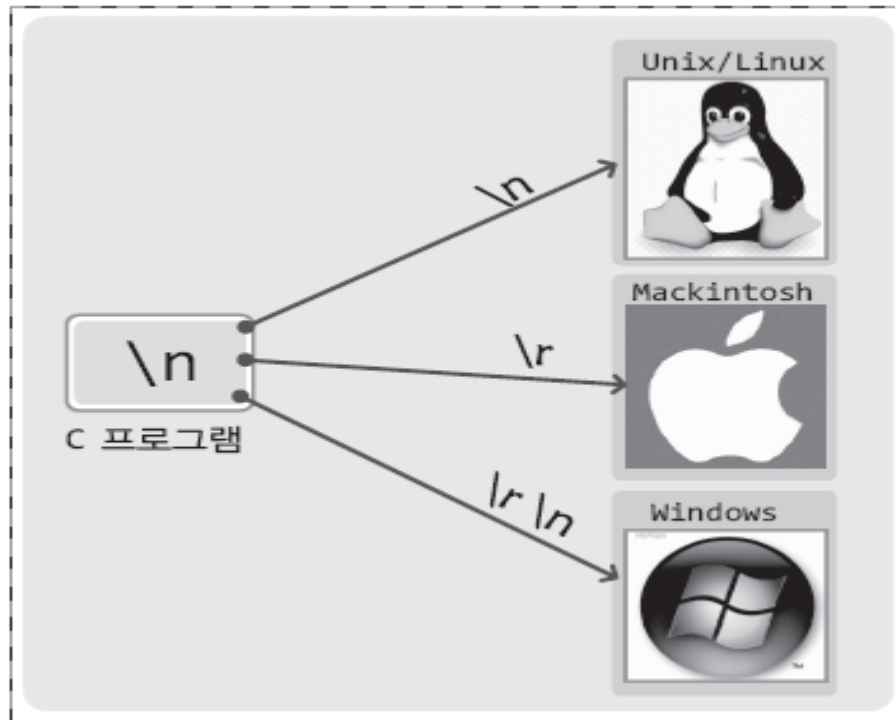
운영체제 별로 개행을 표시하는 방법에는 차이가 있다. 만약에 개행을 `\n`으로 표현하지 않는 운영체제가 있다면?

개행을 `\n`으로 표현하지 않는 운영체제는 `\n`을 전혀 다르게 해석하게 된다.

## 운영체제 별 개행의 표시 방법

- |           |                   |
|-----------|-------------------|
| ▶ Windows | <code>\r\n</code> |
| ▶ Linux   | <code>\n</code>   |
| ▶ Mac     | <code>\r</code>   |

# 텍스트 스트림이 별도로 존재하는 이유



개행 정보를 정확히 저장하기 위해서는 위와 같은 종류의 변환 과정을 거쳐야 한다.

텍스트 모드로 데이터를 입출력 하면 이러한 형태의 변환이 운영체제에 따라서 자동으로 이뤄진다.

## Ch. 21에서 학습한 파일 입출력 함수들

```
int fputc(int c, FILE *stream); /*문자출력*/  
int fgetc(FILE *stream) /* 문자 입력 */  
int fputs(const char *s, FILE *stream);  
char *fgets(char *s, int n, FILE *stream);
```

당시에는 매개변수 stream에 stdin 또는 stdout을 인자로 전달하여 콘솔을 대상으로 입출력을 진행하였지만, 위의 함수들은 FILE 구조체의 포인터를 인자로 전달하여 파일을 대상으로 입출력을 진행할 수 있는 함수들이다.

# 파일 입출력의 예

```
int main(void) {  
    FILE *fp = fopen("simple.txt", "w");  
    if (fp == NULL) {  
        puts("파일 오픈 실패");  
        return -1;  
    }  
    fputc('A', fp); 문자 A와 B가 fp라는 파일에 저장됨  
    fputc('B', fp);  
    fputs("My name is Hong\n", fp);  
    fputs("Your name is Yoon\n", fp);  
    fclose(fp);  
    return 0;  
}
```

# 파일 입출력의 예

```
int main(void) {
    char str[30];
    int ch;
    FILE *fp = fopen("simple.txt", "r");
    if (fp == NULL) {
        puts("파일 오픈 실패");
        return -1;
    }
    ch = fgetc(fp);
    printf("%c\n", ch);
    ch = fgetc(fp);
    printf("%c\n", ch);
}
```

## 파일 입출력의 예

```
fgets(str, sizeof(str), fp); \n을 만날때까지 read  
printf("%s", str);  
fgets(str, sizeof(str), fp); \n을 만날때까지 read  
printf("%s", str);  
fclose(fp);  
return 0;  
}
```



# feof 함수 기반의 파일복사 프로그램

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

```
int feof(FILE *stream);
```

→ 파일의 끝에 도달한 경우 0이 아닌 값 반환

- 파일의 끝을 확인해야 하는 경우, 이 함수가 필요함
- 파일 입력 함수는 오류가 발생하는 경우에도 EOF를 반환
- EOF의 반환 원인을 확인하려면 이 함수를 호출해야 함

# feof 함수 기반의 파일복사 프로그램

```
int main(void) {  
    FILE *src = fopen("simple.txt", "r");  
    FILE *dest = fopen("dst.txt", "w");  
    int ch;  
    if (src == NULL || dest == NULL) {  
        puts("파일오픈 실패!");  
        return -1;  
    }  
}
```

# feof 함수 기반의 파일복사 프로그램

```
while ((ch = fgetc(src)) != EOF)
    fputc(ch, dest);

if (feof(src) != 0)
    puts("파일 복사 완료!");
else
    puts("파일 복사 실패!");
fclose(src);
fclose(dest);
return 0;
}
```

EOF가 반환이  
되면...

feof 함수호출을  
통해서 EOF 반환  
원인을 확인!

# 문자열 단위 파일복사 프로그램

```
int main(void) {  
    FILE* src = fopen("simple.txt", "r");  
    FILE* dest = fopen("dst.txt", "w");  
    char str[30];  
    if (src == NULL || dest == NULL) {  
        puts("파일오픈 실패!");  
        return -1;  
    }  
}
```

# 문자열 단위 파일복사 프로그램

```
while (fgets(str, sizeof(str), src) != NULL)
    fputs(str, dest);
if (feof(src) != 0)
    puts("파일 복사 완료!");
else
    puts("파일 복사 실패!");
fclose(src);
fclose(dest);
return 0;
}
```

EOF가 반환되면...

feof 함수호출을  
통해서 EOF 반환  
원인을 확인!

# 바이너리 데이터의 입출력: fread

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

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

→ 성공 시 전달인자 count, 실패 또는 파일의 끝 도달 시 count보다 작은 값 반환

```
int main(void) {
```

```
    int buf[12];
```

```
    ...
```

```
    fread((void *)buf, sizeof(int), 12, fp);
```

```
    ...
```

sizeof(int) 크기의 데이터 12개를  
fp로부터 읽어 들어서 배열 buf에  
저장하라!

# 바이너리 파일 복사 프로그램

```
int main(void) {  
    FILE* src = fopen("src.bin", "rb");  
    FILE* dest = fopen("dst.bin", "wb");  
    char buf[20];  
    int readCnt;  
    if (src == NULL || dest == NULL) {  
        printf("파일 오픈 실패!");  
        return -1;  
    }  
}
```

# 바이너리 파일 복사 프로그램

```
while (1) {
    readCnt = fread((void*)buf, 1, sizeof(buf), src);
    if (readCnt < sizeof(buf)) { /* 1 */
        if (feof(src) != 0) {
            fwrite((void*) buf, 1, readCnt, dest); /* 2 */
            printf("파일복사 완료");
            break;
        }
        else printf("파일복사 실패");
        break;
    }
    fwrite((void*) buf, 1, sizeof(buf), dest);
}
```



# 바이너리 파일 복사 프로그램

```
fclose(src);  
fclose(dest);  
return 0;  
}
```

1. 파일의 끝에 도달해서 buf를 다 채우지 못한 경우에 참이 됨
2. feof() 함수 호출의 결과가 참이면 파일의 끝에 도달했다는 의미이므로 마지막으로 읽은 데이터를 파일에 저장하고 프로그램을 종료함. feof()함수의 결과가 거짓이면, 에러가 발생한 것으로 가정함.

## 서식에 따른 데이터 입출력: fprintf, fscanf

```
char name[10] = "홍길동";  
char gen = 'M';  
int age = 24;  
fprintf(fp, "%s %c %d", name, gen, age);
```

- ▣ fprintf함수는 printf함수와 사용방법이 유사함
- ▣ 파일을 대상으로 조합이 된 문자열이 출력(저장)됨

```
int main(void) {  
    char name[10];  
    char gen;  
    int age;  
    FILE* fp = fopen("friend.txt", "w");  
    int i;
```

## 서식에 따른 데이터 입출력: fprintf, fscanf

```
for (i = 0; i < 3; i++) {  
    printf("이름 성별 나이 순 입력: ");  
    scanf("%s %c %d", name, &gen, &age);  
    getchar(); /* 버퍼에 남아 있는 \n을 소진 */  
    fprintf(fp, "%s %c %d", name, gen, age);  
}  
fclose(fp);  
return 0;  
}
```

### 실행결과

```
이름 성별 나이 순 입력: 정은영 F 22  
이름 성별 나이 순 입력: 한수정 F 26  
이름 성별 나이 순 입력: 이영호 M 31
```

# 서식에 따른 데이터 입출력: fprintf, fscanf

```
char name[10] = "홍길동";  
char gen = 'M';  
int age = 24;  
fscanf(fp, "%s %c %d", name, &gen, &age);
```

- fscanf함수는 scanf함수와 사용방법이 유사함
- 파일로부터 서식문자의 조합 형태대로 데이터가 입력됨

```
int main(void) {  
    char name[10];  
    char gen;  
    int age;  
    FILE* fp = fopen("friend.txt", "r");  
    int ret;
```

## 서식에 따른 데이터 입출력: fprintf, fscanf

```
while (1) {  
    ret = fscanf(fp, "%s %c %d", name, &gen, &age);  
    if (ret == EOF)  
        break;  
    printf("%s %c %d\n", name, gen, age);  
}  
fclose(fp);  
return 0;  
}
```

실행결과

정은영 F 22

한수정 F 26

이영호 M 31

# Text/Binary의 집합체인 구조체 변수 입출력

- 구조체 변수의 입출력은 생각보다 어렵지 않음
- fread & fwrite 함수로 통째로 입출력을 하면 됨

```
typedef struct fren {  
    char name[10];  
    char gen;  
    int age;  
} Friend;  
  
int main(void) {  
    FILE* fp;  
    Friend myfren1, myfren2;
```

```
/** file write */
fp = fopen("friend.bin", "wb");
printf("이름, 성별, 나이 순 입력: ");
scanf("%s %c %d", myfren1.name, &(myfren1.gen),
      &(myfren1.age));
fwrite((void*) &myfren1, sizeof(myfren1), 1, fp);
fclose(fp);
/** file read */
fp = fopen("friend.bin", "rb");
fread((void*) &myfren2, sizeof(myfren2), 1, fp);
printf("%s %c %d\n", myfren2.name, myfren2.gen,
      myfren2.age);
fclose(fp);
return 0;
}
```

### 실행결과

```
이름, 성별, 나이 순 입력: Jungs M 27
Jungs M 27
```

# 파일 위치 지시자란?

- FILE 구조체의 멤버 중 하나
- read 모드로 열린(open)된 파일 위치 지시자: "어디까지 읽었더라?"에 대한 답
- write 모드로 열린(open)된 파일 위치 지시자: "어디부터 이어서 쓰더라?"에 대한 답
- 즉 Read/Write에 대한 위치 정보를 갖고 있음
- 파일 입출력과 관련있는 fputs, fread, fwrite 같은 함수가 호출될 때마다 파일 위치 지시자의 참조 위치는 변경됨



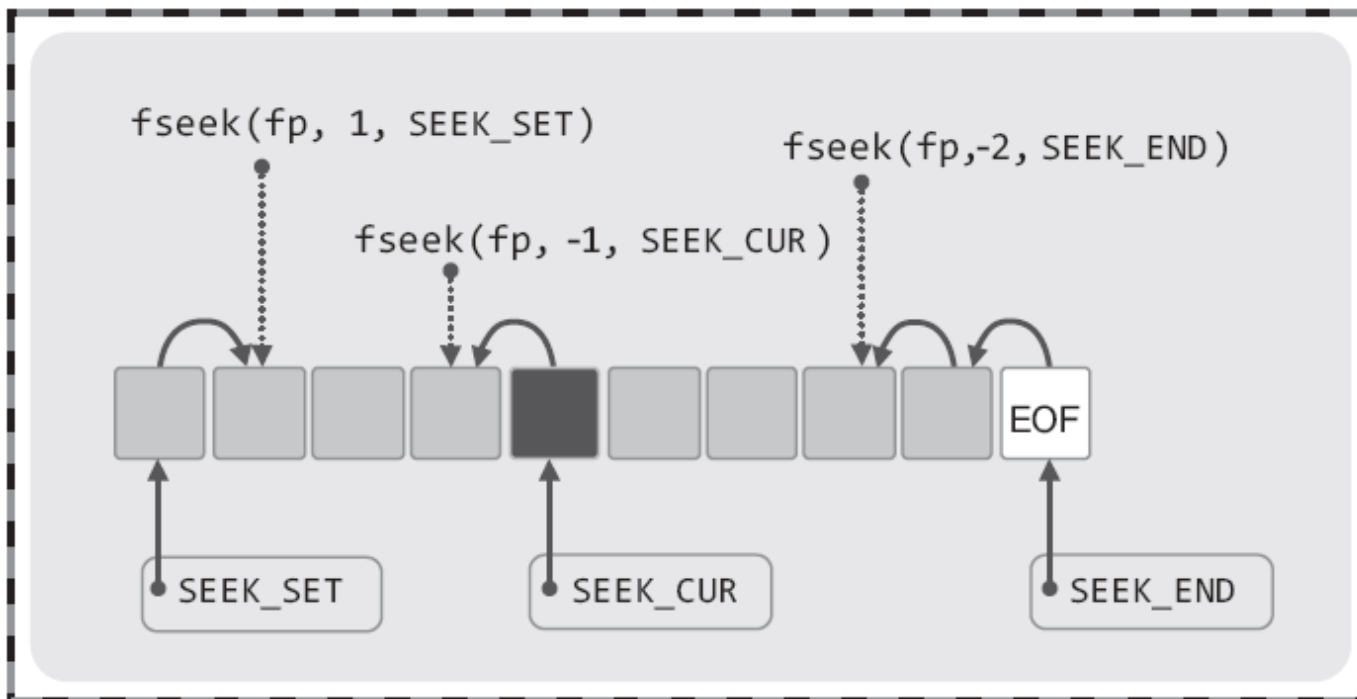
# 파일 위치 지시자의 이동: fseek

## ▣ 파일 위치 지시자의 참조 위치를 변경시키는 함수

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

```
int fseek(FILE *stream, long offset, int wherefrom);
```

→ 성공 시 0, 실패시 0이 아닌 값을 반환



fseek 함수의  
호출결과로  
인한 파일  
위치 지시자의  
이동 결과

# 파일 위치 지시자의 이동: fseek

| 매개변수<br>wherefrom이 ... | 값 | 파일 위치 지시자는 ...    |
|------------------------|---|-------------------|
| SEEK_SET               | 0 | 파일 맨 앞에서부터 이동을 시작 |
| SEEK_CUR               | 1 | 현재 위치에서부터 이동을 시작  |
| SEEK_END               | 2 | 파일 맨 끝에서부터 이동을 시작 |

## fseek 함수의 호출 예

```
int main(void) {  
    /* 파일 생성 */  
    FILE *fp = fopen("test.txt", "w");  
    fputs("123456789", fp);  
    fclose(fp);  
    /* 파일 개방 */  
    fp = fopen("test.txt", "r");  
    /* SEEK_END test */  
    fseek(fp, -2, SEEK_END); 1 2 3 4 5 6 7 8 9 (eof)  
    printf("%c", fgetc(fp)); 1 2 3 4 5 6 7 8 9 (eof)  
    /* SEEK_SET test */  
    fseek(fp, 2, SEEK_SET); 1 2 3 4 5 6 7 8 9 (eof)  
    printf("%c", fgetc(fp)); 1 2 3 4 5 6 7 8 9 (eof)  
}
```

## fseek 함수의 호출 예

```
/* SEEK_CUR test */  
fseek(fp, 2, SEEK_CUR); 1 2 3 4 5 6 7 8 9 (eof)  
printf("%c", fgetc(fp)); 1 2 3 4 5 6 7 8 9 (eof)  
fclose(fp);  
return 0;  
}
```

실행결과

836

# 현재 파일 위치 지시자의 위치는?: ftell

▣ 현재 파일 위치자의 위치 정보를 반환하는 함수!

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

```
int ftell(FILE *stream);
```

→ 파일 위치 지시자의 위치 정보 반환

```
int main(void) {
```

```
    long fpos;
```

```
    int i;
```

```
    /* 파일 생성 */
```

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

```
    fputs("1234-", fp);
```

```
    fclose(fp);
```

# 현재 파일 위치 지시자의 위치는?: ftell

```
/* 파일 개방 */
fp = fopen("test.txt", "r");
for (i = 0; i < 4; i++) {
    printf("%c", fgetc(fp));
    fpos = ftell(fp); /* 현재 위치 저장 */
    fseek(fp, -1, SEEK_END); /* 맨 뒤로 이동 */
    printf("%c", fgetc(fp));
    /* 저장해놓은 위치 복원 */
    fseek(fp, fpos, SEEK_SET);
}
fclose(fp);
return 0;
}
```

실행결과

1-2-3-4-

# 정리

---

- 기본 입출력
- file open mode
- 파일 입출력 함수
- 서식이 있는 입출력
- 파일 읽기/쓰기 위치 변경