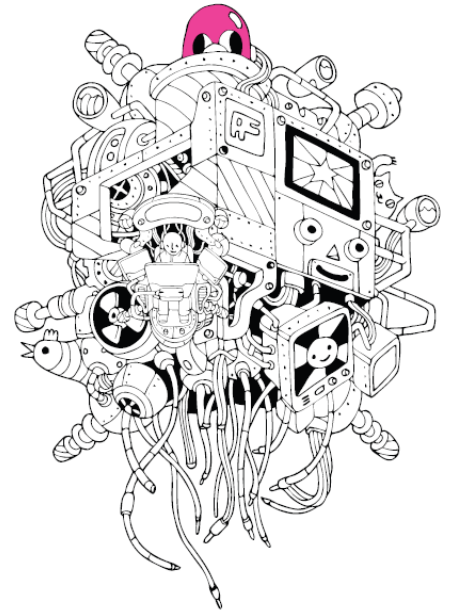


윤성우의 열혈 C 프로그래밍



윤성우 저 열혈강의 C 프로그래밍 개정판

Chapter 24. 파일 입출력

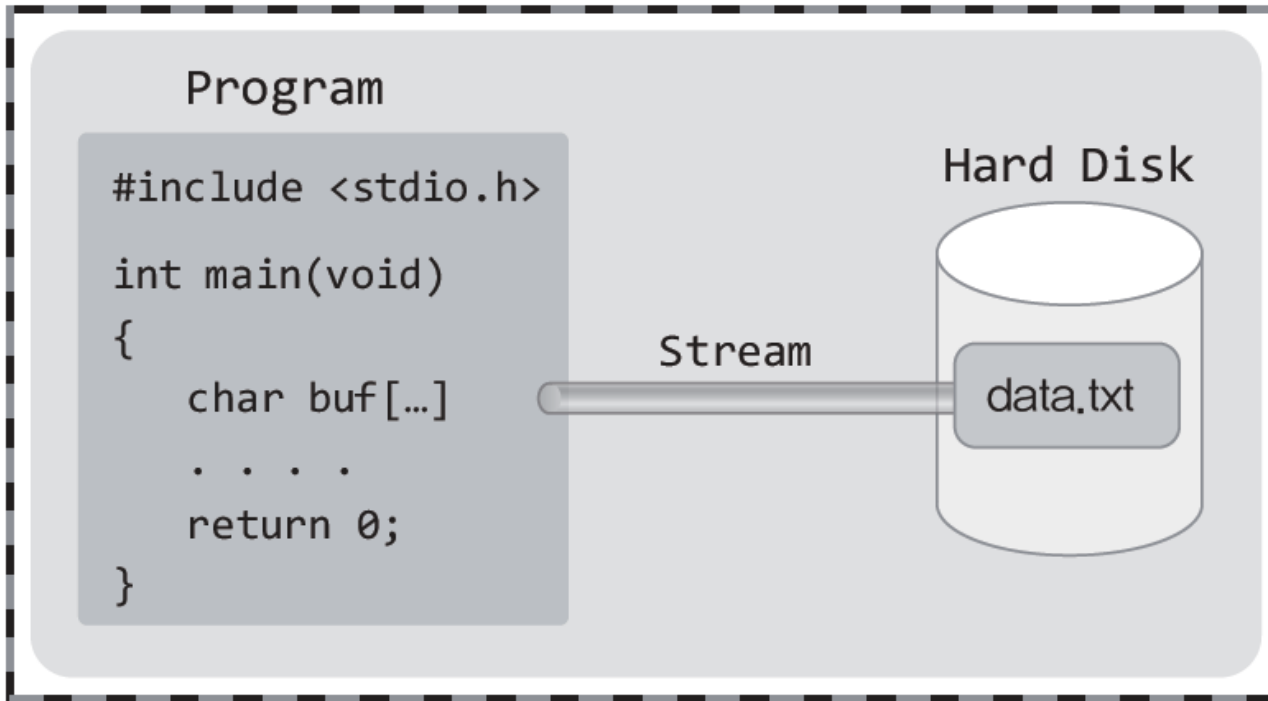
윤성우의 열혈 C 프로그래밍



Chapter 24-1. 파일과 스트림 그리고
기본적인 파일의 입출력

윤성우 저 열혈강의 C 프로그래밍 개정판

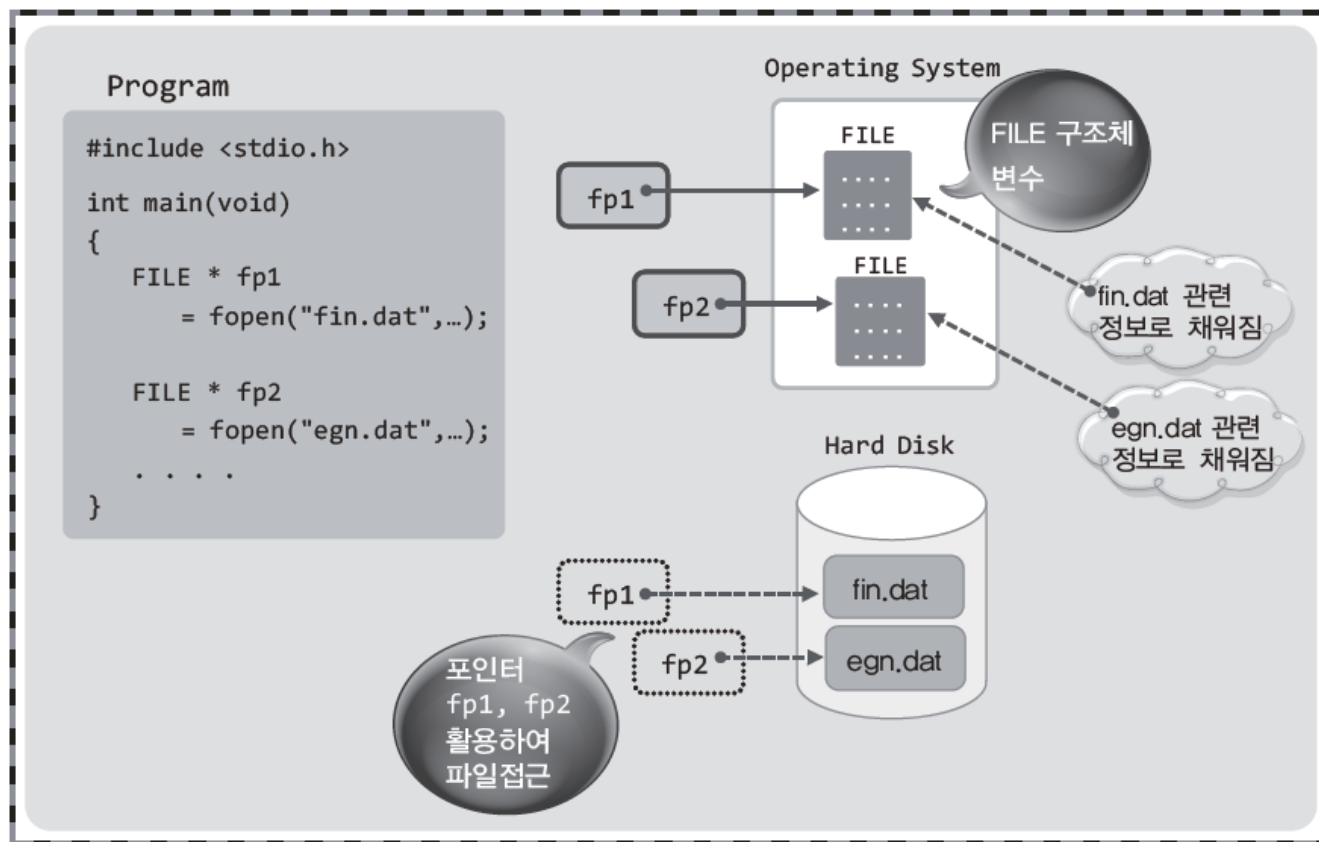
파일에 저장되어 있는 데이터를 읽고 싶어요.



콘솔 입출력과 마찬가지로 파일로부터의 데이터 입출력을 위해서는 **스트림**이 형성되어야 한다.

파일과의 스트림 형성은 데이터 입출력의 기본이다.

fopen 함수를 통한 스트림의 형성과 FILE 구조체



fopen 함수호출 시 생성되는 **FILE 구조체 변수**와 이를 참조하는 **FILE 구조체 포인터 변수**의 관계를 이해하자!

fopen 함수호출의 결과

스트림을 형성할 파일의 이름

형성할 스트림의 종류

```
#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 생성된다.
- 생성된 FILE 구조체 변수에는 파일에 대한 정보가 담긴다.
- FILE 구조체의 포인터는 사실상 파일을 가리키는 '지시자'의 역할을 한다.

fopen 함수가 파일과의 스트림 형성을 요청하는 기능의 함수이다.

출력 스트림의 생성

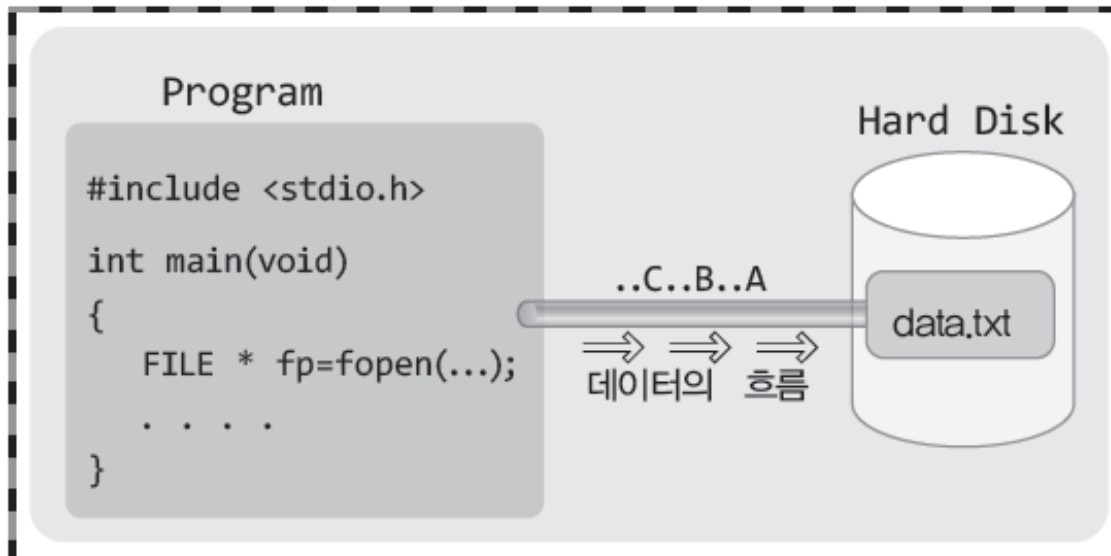
```
FILE * fp = fopen("data.txt", "wt");
```

← "wt"에는 출력 스트림의 의미가 담겨있다.

“파일 data.txt와 스트림을 형성하되 wt 모드로 스트림을 형성해라!”



출력 스트림의 형성 결과



포인터 변수 *fp*에 저장된 값이 *data.txt*의 스트림에 데이터를 전송하는 도구가 된다.

입력 스트림의 생성

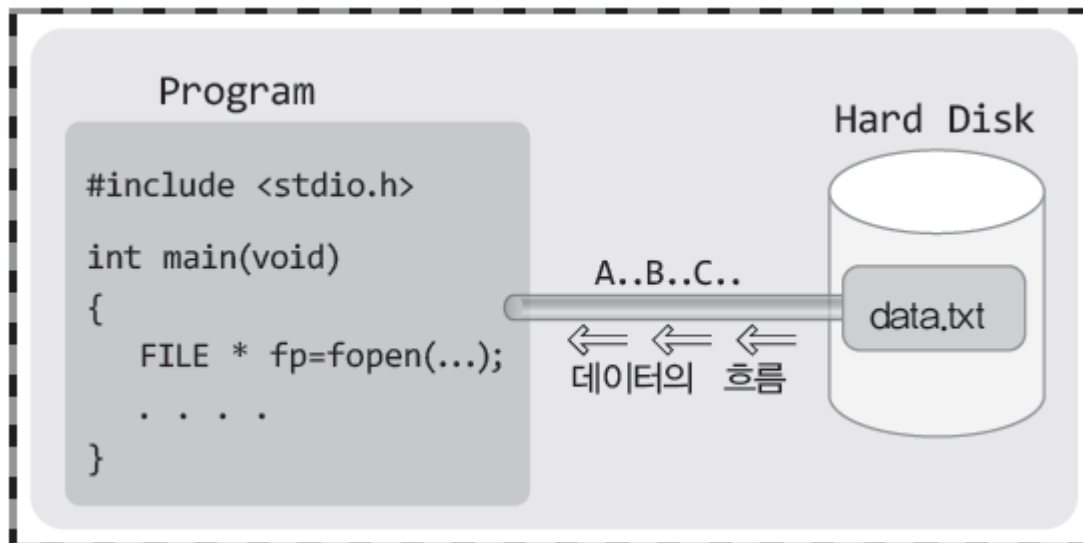
```
FILE * fp = fopen("data.txt", "rt");
```

"rt"에는 입력 스트림의 의미가 담겨있다.

“파일 data.txt와 스트림을 형성하되 rt 모드로 스트림을 형성해라!”



입력 스트림의 형성 결과



포인터 변수 *fp*에 저장된 값이 *data.txt*의 스트림으로부터 데이터를 수신하는 도구가 된다.

파일에 데이터를 써봅시다.

현재 디렉터리에 저장된 파일 data.txt를 찾는다.

```
int main(void)
{
    FILE * fp=fopen("data.txt", "wt");
    if(fp==NULL) {
        puts("파일오픈 실패!");
        return -1;    // 비정상적 종료를 의미하기 위해서 -1을 반환
    }

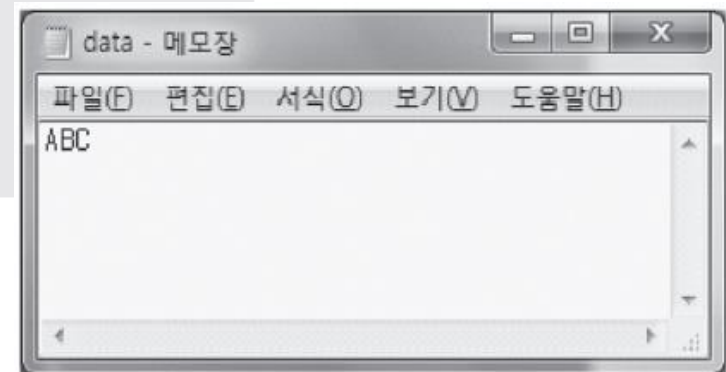
    fputc('A', fp);
    fputc('B', fp);
    fputc('C', fp);
    fclose(fp);    // 스트림의 종료
    return 0;
}
```

현재 디렉터리는 실행파일이 저장된 디렉터리이거나
프로젝트 파일이 저장된 디렉터리이다!

← 문자 A를 fp가 가리키는 파일에 저장해라!

FILE * fp = fopen("C:\\Project\\data.txt", "wt");

fopen 함수호출 시 경로를 완전히 명시할 수도 있다.



메모장으로 파일을 열어서 확인해 본다.

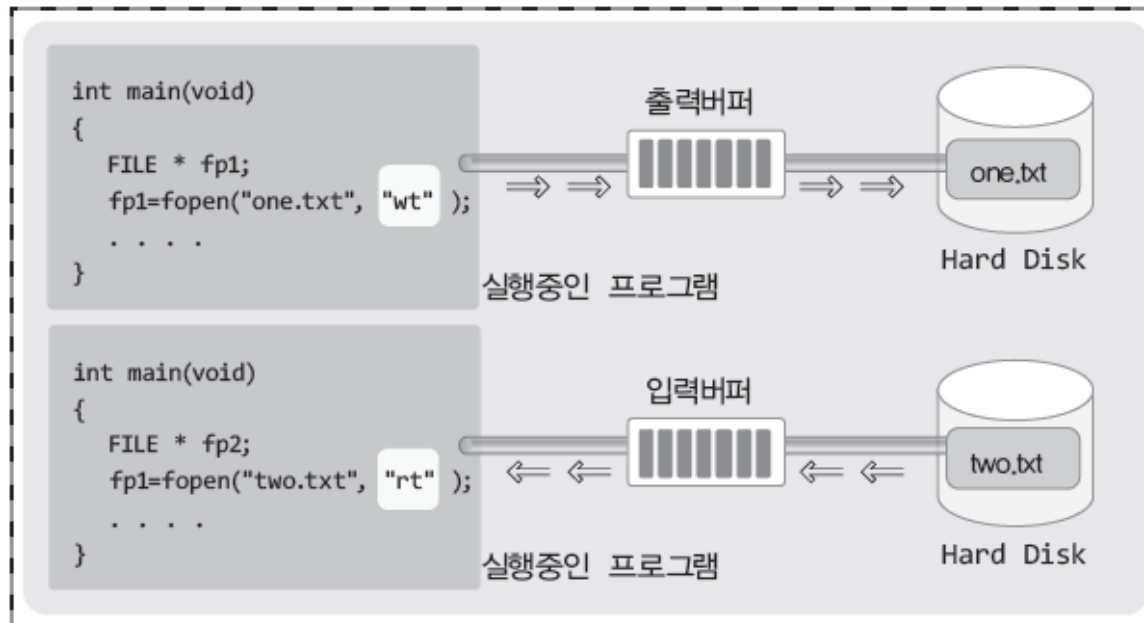
스트림의 소멸을 요청하는 fclose 함수

```
#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



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

Ch21에서 호출한 적 있는 fflush 함수

```
#include <stdio.h>
int fflush(FILE * stream);
```

➔ 함수호출 성공 시 0, 실패 시 EOF 반환

콘솔 대상으로 *fflush* 함수를 설명한바 있다.

대상이 파일로 바뀌었을 뿐 달라지는 것은 없다.

- 출력버퍼를 비운다는 것은 출력버퍼에 저장된 데이터를 목적지로 전송한다는 의미
- 입력버퍼를 비운다는 것은 입력버퍼에 저장된 데이터를 소멸시킨다는 의미
- fflush 함수는 출력버퍼를 비우는 함수이다.
- fflush 함수는 입력버퍼를 대상으로 호출할 수 없다.

```
int main(void)
{
    FILE * fp = fopen("data.txt", "wt");
    . . .
    fflush(fp); // 출력 버퍼를 비우라는 요청!
    . . .
}
```

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

그렇다면 파일의 입력버퍼는 어떻게 비우는가?

이를 위한 별도의 함수가 정의되어 있는가?

입력 버퍼를 비우는 방법

```
void main()
{
    char str[32];
    char ch;

    scanf("%s", str);
    
    scanf("%c", &ch);
}
```

← *ch에 개행문자가 들어가기 때문에 입력버퍼를 비워줘야 함*

```
void main()
{
    char str[32];
    char ch;

    scanf("%s", str);
    getchar();

    scanf("%c", &ch);
}
```

```
void main()
{
    char str[32];
    char ch;

    fgets(str, sizeof(str), stdin);
    str[strlen(str) - 1] = '\0';
    scanf("%c", &ch);
}
```

```
void main()
{
    char str[32];
    char ch;

    scanf("%s", str);
    while (getchar() != '\0');

    scanf("%c", &ch);
}
```



파일로부터 데이터를 읽어 봅시다.

```
int main(void)
{
    int ch, i;
    FILE * fp=fopen("data.txt", "rt");
    if(fp==NULL) {
        puts("파일오픈 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!*

실행결과

A 이전에 문자가 써진 순서대로 읽힌다!

B

C

윤성우의 열혈 C 프로그래밍



Chapter 24-2. 파일의 개방 모드

윤성우 저 열혈강의 C 프로그래밍 개정판

스트림의 구분 기준 두 가지(Basic)

• 기준1

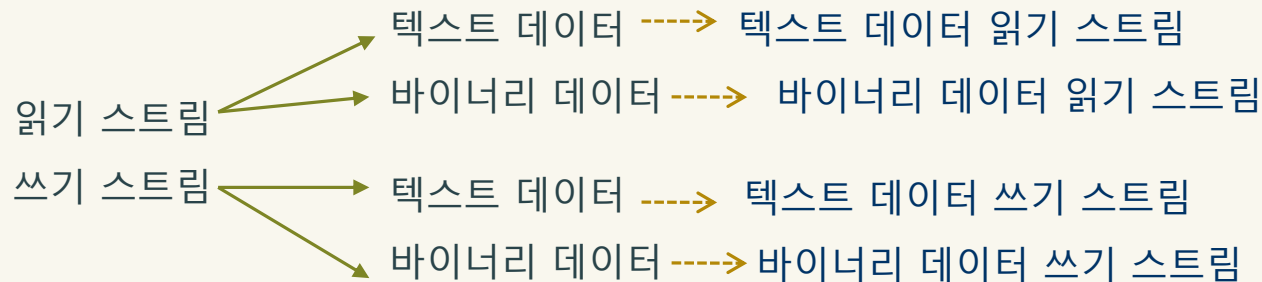
읽기 위한 스트림이냐? 쓰기 위한 스트림이냐?

파일에 데이터를 쓰는데 사용하는 스트림과 데이터를 읽는데 사용하는 스트림은 구분이 된다.

• 기준2

텍스트 데이터를 위한 스트림이냐? 바이너리 데이터를 위한 스트림이냐?

추력의 대상이 되는 데이터의 종류에 따라서 스트림은 두 가지로 나뉜다.



기본적인 스트림의 구분!
그러나 실제로는 더 세분화!

스트림을 구분하는 기준1: Read or Write

√ 스트림의 성격은 R/W를 기준으로 다음과 같이 세분화 된다.

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, 쓰기가 모두 가능한 스트림의 형성을 의미한다.

➔ 모드의 a는 쓰기가 가능한 스트림을 의미하되 여기서 말하는 쓰기는 덧붙여 쓰기이다.



스트림을 구분하는 기준2: 텍스트 모드, 바이너리 모드

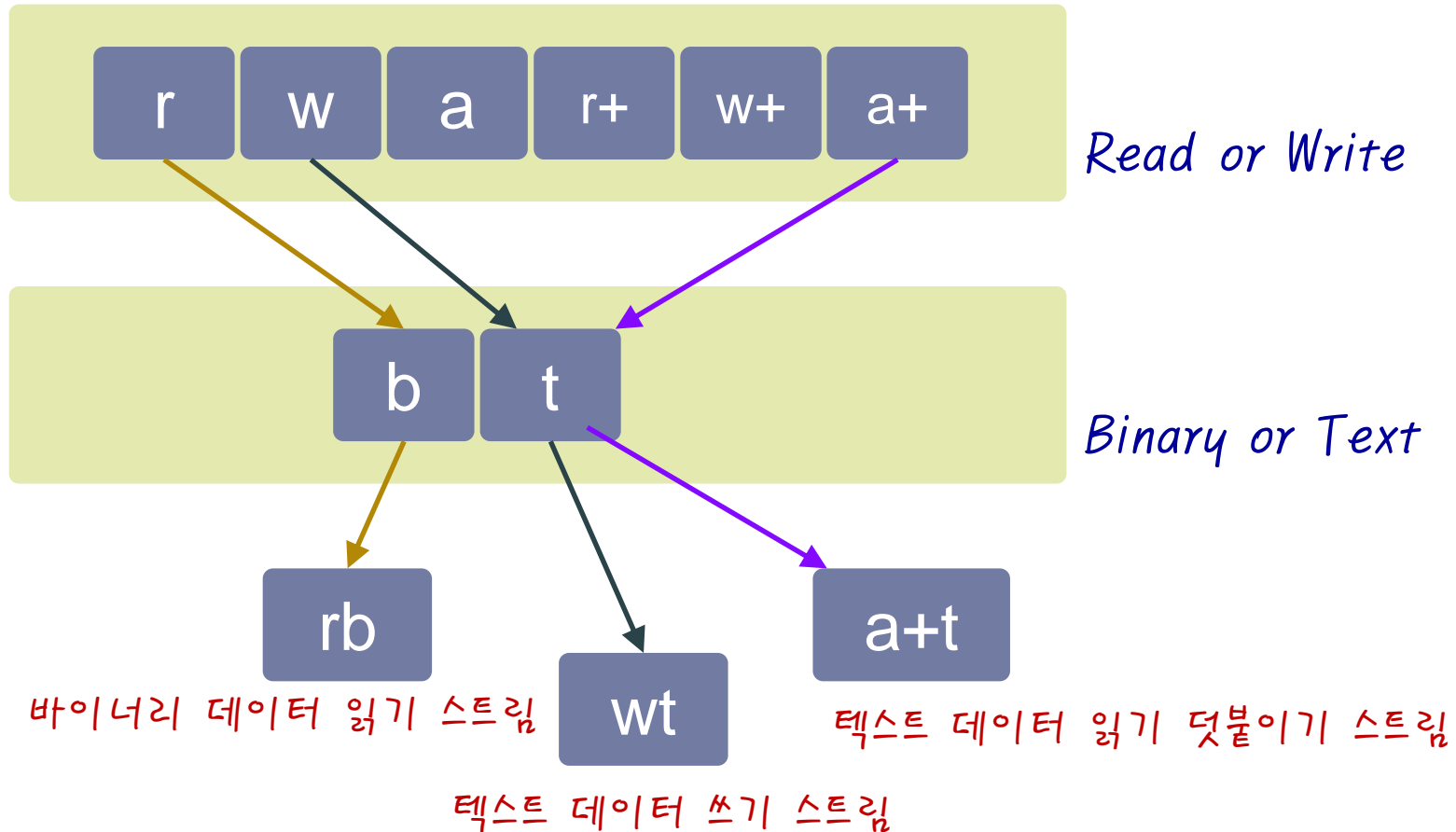
√ 스트림의 성격은 데이터의 종류에 따라서 다음과 같이 두 가지로 나뉜다.

- ▶ 텍스트 모드 스트림 (**t**) : 문자 데이터를 저장하는 스트림
- ▶ 바이너리 모드 스트림 (**b**) : 바이너리 데이터를 저장하는 스트림

√ 문자 데이터와 바이너리 데이터

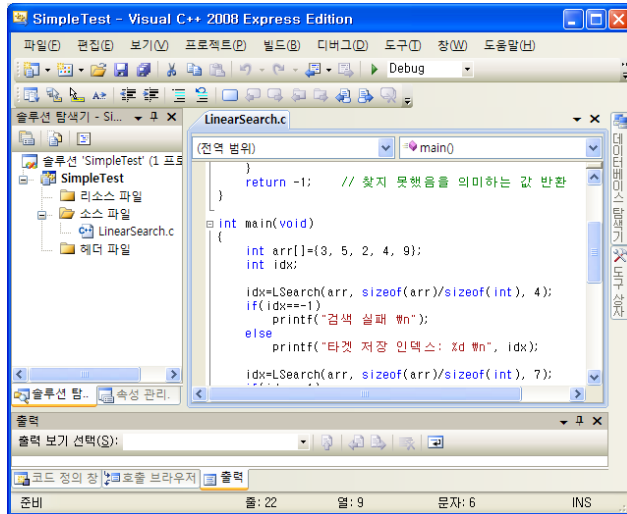
- ▶ 문자 데이터 : 사람이 인식할 수 있는 유형의 문자로 이뤄진 데이터
 - 파일에 저장된 문자 데이터는 Windows의 메모장으로 열어서 문자 확인이 가능
 - 예 : 도서목록, 물품가격, 전화번호, 주민등록번호
- ▶ 바이너리 데이터 : 컴퓨터가 인식할 수 있는 유형의 데이터
 - 메모장과 같은 편집기로는 그 내용이 의미하는 바를 이해할 수 없다.
 - 예 : 음원 및 영상 파일, 그래픽 디자인 프로그램에 의해 저장된 디자인 파일

파일의 개방모드 조합!



t도 b도 붙지 않으면 텍스트 모드로 파일 개방

텍스트 스트림이 별도로 존재하는 이유1



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



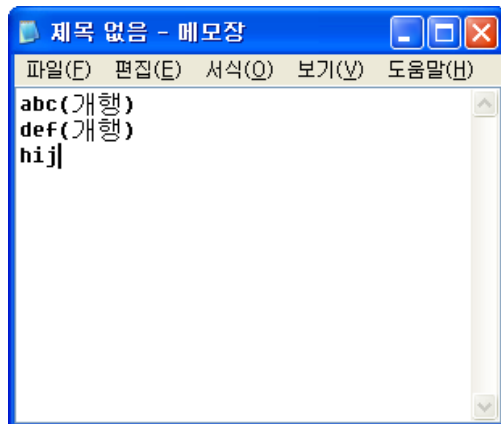
개행 정보로 저장된 `\n`은 문제가 되지 않을까?

text.txt

텍스트 스트림이 별도로 존재하는 이유2

text.txt

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

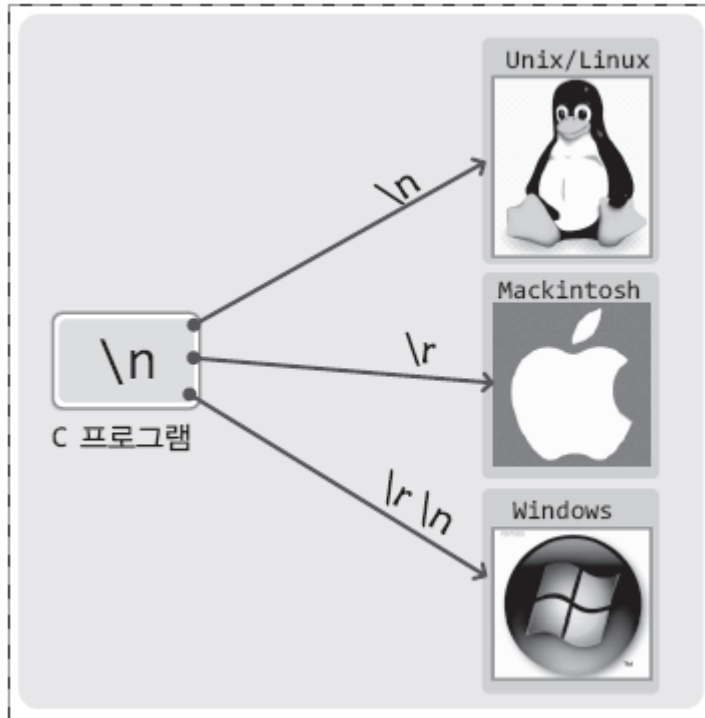


개행을 `\n`으로 표현하지 않는 운영체제는 `\n`을
전혀 다르게 해석하게 된다.

운영체제 별 개행의 표시 방법

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

텍스트 스트림이 별도로 존재하는 이유3



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

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

A collection of 15 cartoon icons representing various objects and animals. The icons include: a planet with a ring and a face, a key, a ghost, a robot, a laptop, a cat, a fish, a die, a TV, a planet, a cat, a fish, a die, a TV, a planet, a cat, a fish, a die, a TV.

윤성우 저 열혈강의 C 프로그래밍 개정판

Chapter 21에서 학습한 파일 입출력 함수들

텍스트 데이터 입출력 함수들

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

당시에는 매개변수 *stream*에 *stdin* 또는 *stdout*을 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", "wt");
    if(fp==NULL) {
        puts("파일오픈 실패!");
        return -1;
    }

    fputc('A', fp); 문자 A와 B가
    fputc('B', fp); fp가 가리키는 파일에 저장
    fputs("My name is Hong \n", fp);
    fputs("Your name is Yoon \n", fp);
    fclose(fp); 두 개의 문자열이 fp가 가리
    return 0; 키는 파일에 저장
}
```

파일에 저장된 문자열의 끝에는 널이 존재하지 않는다.
때문에 파일을 대상으로 문자열을 입출력 할 때에는 개
행을 의미하는 \n을 문자열의 마지막에 넣어줘야 한
다. \n을 기준으로 문자열을 구분하기 때문이다.

write 순서대로 read해야 한다!

```
int main(void)
{
    char str[30];
    int ch;
    FILE * fp=fopen("simple.txt", "rt");
    if(fp==NULL) {
        puts("파일오픈 실패!");
        return -1;
    }

    ch=fgetc(fp);
    printf("%c \n", ch);
    ch=fgetc(fp);
    printf("%c \n", ch);

    fgets(str, sizeof(str), fp);
    printf("%s", str); \n을 만날때까지 read
    fgets(str, sizeof(str), fp);
    printf("%s", str); \n을 만날때까지 read

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

A 실행결과
B
My name is Hong
Your name is Yoon

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 필요하다. 파일 입력 함수는 오류가 발생하는 경우에도 EOF를 반환한다. 따라서 EOF의 반환원인을 확인하려면 이 함수를 호출해야 한다.

feof 함수호출을 통해서
EOF 반환 원인을 확인!

```
int main(void) 문자 단위 파일복사 프로그램
{
    FILE * src=fopen("src.txt", "rt");
    FILE * des=fopen("dst.txt", "wt");
    int ch;
    if(src==NULL || des==NULL) {
        puts("파일오픈 실패!");
        return -1;
    }
    while((ch=fgetc(src))!=EOF)
        fputc(ch, des);
    if(feof(src)!=0)
        puts("파일복사 완료!");
    else
        puts("파일복사 실패!");
    fclose(src);
    fclose(des);
    return 0;
}
```

EOF가 반환이 되면...

문자열 단위 파일복사 프로그램

```
int main(void)
{
    FILE * src=fopen("src.txt", "rt");
    FILE * des=fopen("des.txt", "wt");
    char str[20];

    if(src==NULL || des==NULL) {
        puts("파일오픈 실패!");
        return -1;
    }

    while(fgets(str, sizeof(str), src)!=NULL)
        fputs(str, des);

    if(feof(src)!=0)
        puts("파일복사 완료!");
    else
        puts("파일복사 실패!");

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

EOF가 반환이 되면...

문자 단위로 복사를 진행하느냐 문자열 단위로
복사를 진행하느냐의 차이만 있을 뿐!

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 들여서 배열 buf에 저장하라!*



바이너리 데이터의 입출력: fwrite

```
#include <stdio.h>
size_t fwrite(const void * buffer, size_t size, size_t count, FILE * stream);
```

➔ 성공 시 전달인자 count, 실패 시 count보다 작은 값 반환

```
int main(void)
{
    int buf[7]={1, 2, 3, 4, 5, 6, 7};
    . . . .
    fwrite((void*)buf, sizeof(int), 7, fp);
    . . . .
```

*sizeof(int) 크기의 데이터 7개를 buf로부터
읽어서 fp에 저장해라!*



바이너리 파일 복사 프로그램

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

1.
파일의 끝에 도달해서 *buf*를 다 채우지 못한
경우에 참이 된다!

2.
feof 함수호출의 결과가 참이면 파일의 끝에
도달했다는 의미이므로 마지막으로 읽은 데이
터를 파일에 저장하고 프로그램을 종료한다!

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

        break;
    }
    fwrite((void*)buf, 1, sizeof(buf), des);
}
fclose(src);
fclose(des);
return 0;
}
```

윤성우의 열혈 C 프로그래밍



Chapter 24-4. 텍스트 데이터와 바이너리 데이터를 동시에 입출력 하기

윤성우 저 열혈강의 C 프로그래밍 개정판

서식에 따른 데이터 입출력: fprintf, fscanf

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

fprintf 함수를 이용하면 어떻게 텍스트 & 바이너리 데이터를 동시에 출력할 수 있을까?

fprintf 함수는 *printf* 함수와 그 사용방법이 매우 유사하다. 다만 *fp*를 대상으로 조합이 된 문자열이 출력(저장)될 뿐이다.

```
char name[10];
char sex;
int age;
fscanf(fp, "%s %c %d", name, &sex, &age);
```

fscanf 함수를 이용하면 어떻게 텍스트 & 바이너리 데이터를 동시에 입력할 수 있을까?

sprintf 함수는 *printf* 함수와 그 사용방법이 매우 유사하다. 다만 *fp*를 대상으로 서식문자의 조합 형태대로 데이터가 입력될 뿐이다.

fprintf & fscanf 관련 예제

```
int main(void)
{
    char name[10];
    char sex;
    int age;

    FILE * fp=fopen("friend.txt", "wt");
    int i;
    for(i=0; i<3; i++)
    {
        printf("이름 성별 나이 순 입력: ");
        scanf("%s %c %d", name, &sex, &age);
        getchar(); // 버퍼에 남아있는 \n의 소멸을 위해서
        fprintf(fp, "%s %c %d", name, sex, age);
    }
    fclose(fp);
    return 0;
}
```

저장하는 데이터가 문자열이므로
텍스트 모드로 개방한다!

정은영 F 22
한수정 F 26
이영호 M 31

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

실행결과

```
int main(void)
{
    char name[10];
    char sex;
    int age;

    FILE * fp=fopen("friend.txt", "rt");
    int ret;
    while(1)
    {
        ret=fscanf(fp, "%s %c %d", name, &sex, &age);
        if(ret==EOF)
            break;
        printf("%s %c %d \n", name, sex, age);
    }
    fclose(fp);
    return 0;
}
```

Text/Binary의 집합체인 구조체 변수 입출력

```
typedef struct fren
{
    char name[10];
    char sex;
    int age;
} Friend;
```

```
int main(void)
{
    FILE * fp;
    Friend myfren1;
    Friend myfren2;

    /*** file write ***/
    fp=fopen("friend.bin", "wb");
    printf("이름, 성별, 나이 순 입력: ");
    scanf("%s %c %d", myfren1.name, &(myfren1.sex), &(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.sex, myfren2.age);
    fclose(fp);
    return 0;
}
```

바이너리 모드로 통째로
구조체 변수를 저장

바이너리 모드로 통째로
구조체 변수를 복원

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

실행결과

구조체 변수의 입출력은 생각보다 어렵지 않다.

fread & fwrite 함수 기반으로 통째로 입출력 하면 된다.

윤성우의 열혈 C 프로그래밍



Chapter 24-5. 임의 접근을 위한 '파일
위치 지시자'의 이동

윤성우 저 열혈강의 C 프로그래밍 개정판

파일 위치 지시자란?

- FILE 구조체의 멤버 중 하나.
- read 모드로 오픈 된 파일 위치 지시자: "어디까지 읽었더라?"에 대한 답
- write 모드로 오픈 된 파일 위치 지시자: "어디부터 이어서 쓰더라?"에 대한 답
- 즉, Read/Write에 대한 위치 정보를 갖고 있다.

따라서 파일 입출력과 관련이 있는 *fputs*, *fread*, *fwrite*와 같은 함수가 호출될 때마다 파일 위치 지시자의 참조 위치는 변경이 된다.



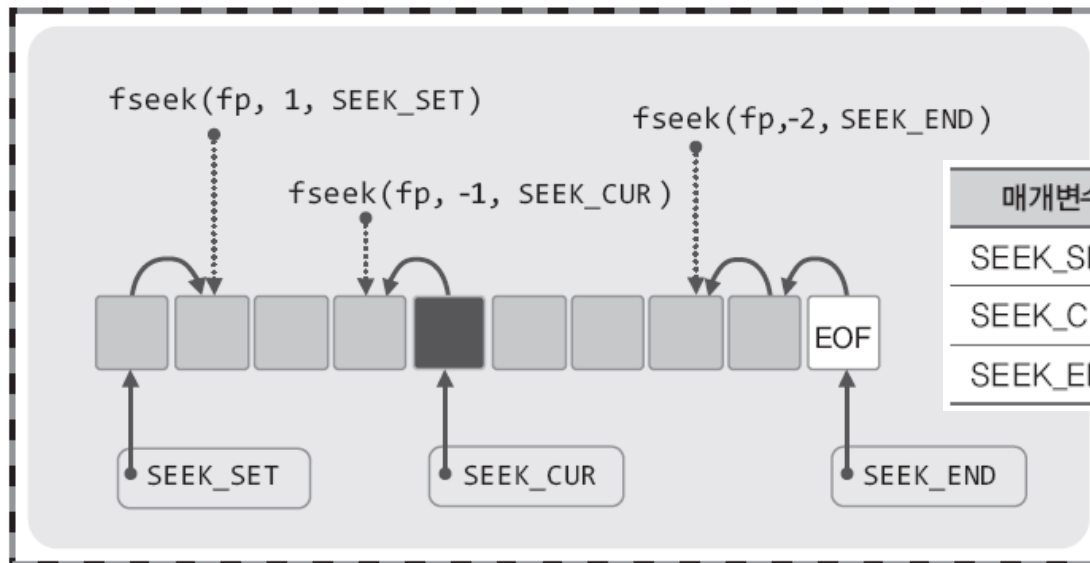
파일 위치 지시자의 이동: fseek

```
#include <stdio.h>
int fseek(FILE * stream, long offset, int wherefrom);
```

➔ 성공 시 0, 실패 시 0이 아닌 값을 반환

파일 위치 지시자의

참조 위치를 변경시키는 함수



매개변수 wherefrom 이...	파일 위치 지시자는...
SEEK_SET(0) 이라면	파일 맨 앞에서부터 이동을 시작
SEEK_CUR(1) 이라면	현재 위치에서부터 이동을 시작
SEEK_END(2) 이라면	파일 맨 끝에서부터 이동을 시작

fseek 함수의 호출결과로 인한 파일 위치 지시자의 이동 결과

fseek 함수의 호출의 예

```
int main(void)
{
    /* 파일생성 */
    FILE * fp=fopen("text.txt", "wt");
    fputs("123456789", fp);
    fclose(fp);

    /* 파일개방 */
    fp=fopen("text.txt", "rt");

    /* SEEK_END test */
    fseek(fp, -2, SEEK_END);
    putchar(fgetc(fp));

    /* SEEK_SET test */
    fseek(fp, 2, SEEK_SET);
    putchar(fgetc(fp));

    /* SEEK_CUR test */
    fseek(fp, 2, SEEK_CUR);
    putchar(fgetc(fp));

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

/ 2 3 4 5 6 7 8 9 e eof)

/ 2 3 4 5 6 7 8 9 e eof)

/ 2 3 4 5 6 7 8 9 e eof)

/ 2 3 4 5 6 7 8 9 e eof)

/ 2 3 4 5 6 7 8 9 e eof)

/ 2 3 4 5 6 7 8 9 e eof)

실행결과

836

현재 파일 위치 지시자의 위치는?: ftell

```
#include <stdio.h>
long ftell(FILE * stream);
```

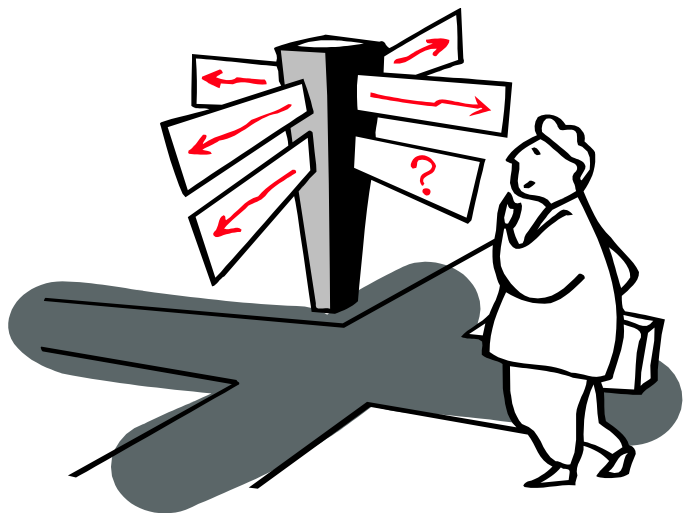
➔ 파일 위치 지시자의 위치 정보 반환

현재 파일 위치자의 위치 정보를 반환하는 함수!

```
int main(void)
{
    long fpos;
    int i;
    /* 파일생성 */
    FILE * fp=fopen("text.txt", "wt");
    fputs("1234-", fp);
    fclose(fp);
    /* 파일개방 */
    fp=fopen("text.txt", "rt");
    for(i=0; i<4; i++)
    {
        putchar(fgetc(fp));
        fpos=ftell(fp); 현재 위치 저장
        fseek(fp, -1, SEEK_END);
        putchar(fgetc(fp));
        fseek(fp, fpos, SEEK_SET);
    } 저장해 놓은 위치 복원
    fclose(fp);
    return 0;
}
```

실행결과

1-2-3-4-



Chapter 24가 끝났습니다. 질문 있으신지요?