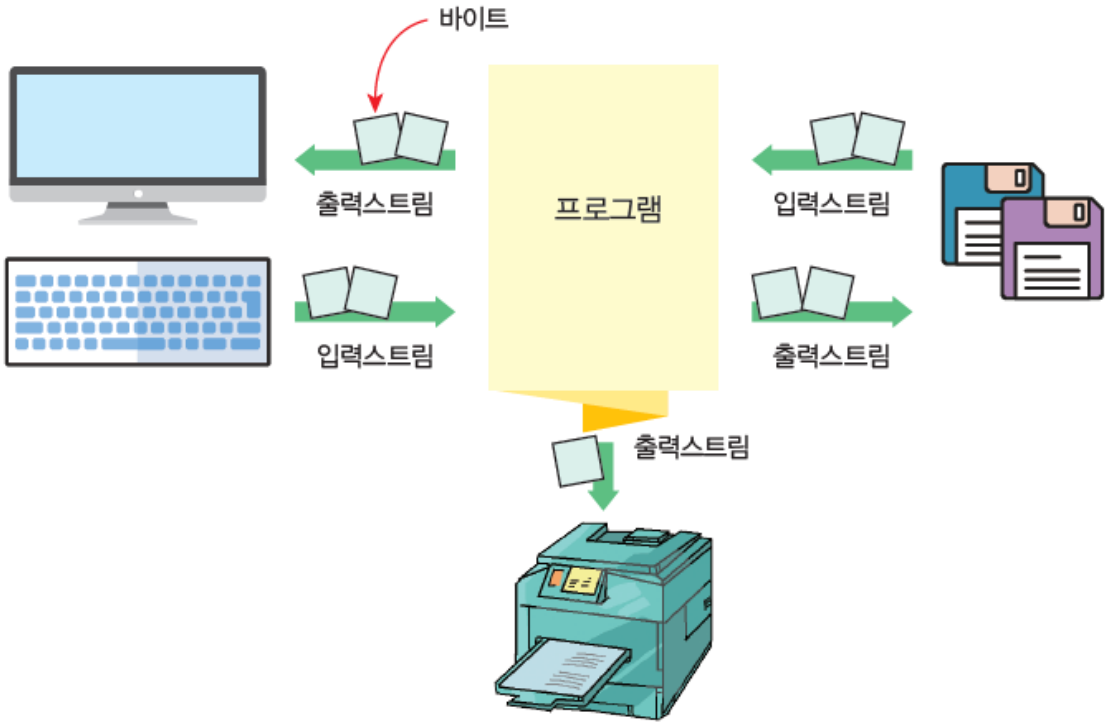
제15장 파일 입출력

이번 장에서 학습할 내용

- 스트림의 개념
- 표준 입출력
- 파일 입출력
- 입출력 관련 함수

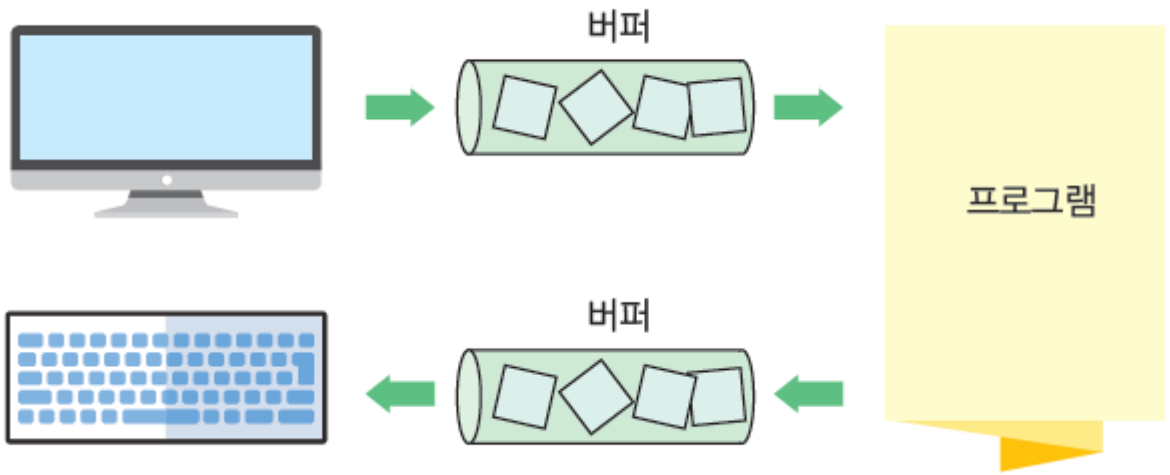
스트림의 개념

- 스트림(stream): 입력과 출력을 바이트(byte)들의 흐름으로 생각하는 것



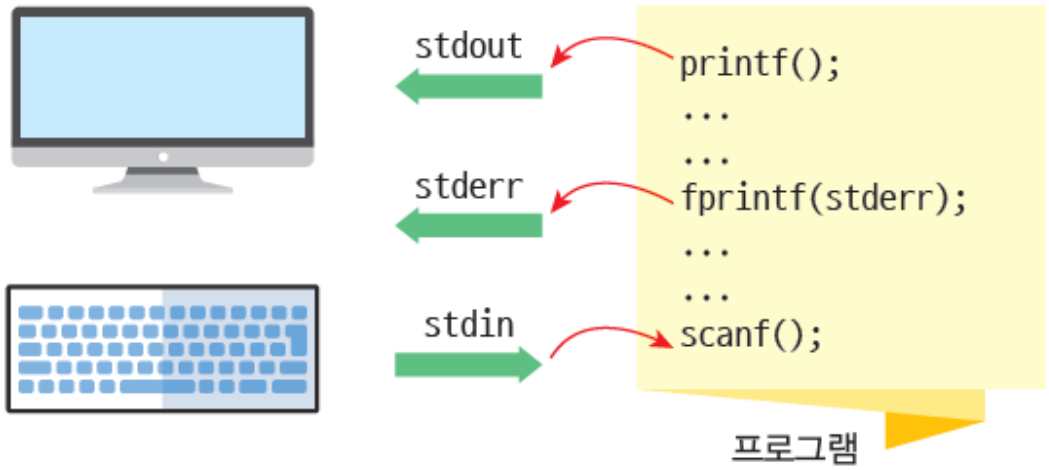
스트림과 버퍼

- 스트림에는 기본적으로 버퍼가 포함되어 있다.



표준 입출력 스트림

이름	스트림	연결 장치
stdin	표준 입력 스트림	키보드
stdout	표준 출력 스트림	모니터의 화면
stderr	표준 오류 스트림	모니터의 화면



입출력 함수의 분류

형식 \ 스트림	표준 스트림	일반 스트림	설명
형식이 없는 입출력 (문자 형태)	getchar()	fgetc(FILE *f,...)	문자 입력 함수
	putchar()	fputc(FILE *f,...)	문자 출력 함수
	gets_s()	fgets(FILE *f,...)	문자열 입력 함수
	puts()	fputs(FILE *f,...)	문자열 출력 함수
형식이 있는 입출력 (정수, 실수...)	printf()	fprintf(FILE *f,...)	형식화된 출력 함수
	scanf()	fscanf(FILE *f,...)	형식화된 입력 함수

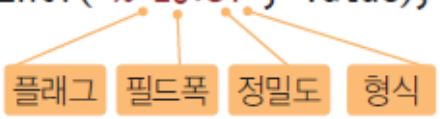
중간 점검

1. C에서의 모든 입력과 출력을 _____형식으로 처리된다.
2. 스트림은 모든 입력과 출력을 _____들의 흐름으로 간주한다.
3. 스트림의 최대 장점은 _____이다.
4. 입력을 위한 표준적인 스트림은 _____이고 기본적으로 _____ 장치와 연결된다.
5. 출력을 위한 표준적인 스트림은 _____이고 기본적으로 _____ 장치와 연결된다.

printf()를 이용한 출력

Syntax: printf()

예 printf("%-10.3f", value);



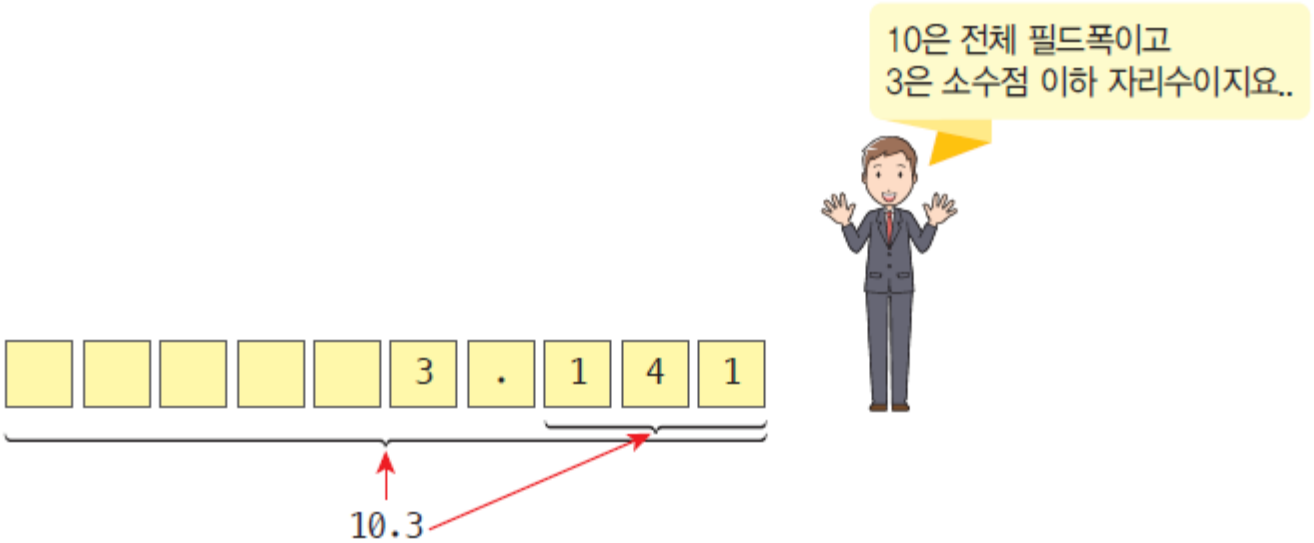
플래그

기호	의미	기본값
-	출력 필드에서 출력값을 왼쪽 정렬한다.	오른쪽 정렬된다.
+	결과 값을 출력할 때 항상 +와 -의 부호를 붙인다.	음수일 때만 - 부호를 붙인다.
0	출력값 앞에 공백 문자 대신에 0으로 채운다. -와 0이 동시에 있으면 0은 무시된다. 만약 정수 출력의 경우, 정밀도가 지정되면 역시 0은 무시된다(예를 들어서 %08.5).	채우지 않는다.
blank(' ')	출력값 앞에 양수나 영인 경우에는 부호대신 공백을 출력한다. 음수일 때는 -가 붙여진다. + 플래그가 있으면 무시된다.	공백을 출력하지 않는다.
#	8진수 출력 시에는 출력값 앞에 0을 붙이고 16진수 출력 시에는 0x를 붙인다.	붙이지 않는다.

플래그

출력 문장	출력 결과	설명
printf("%d", 123);	1 2 3	정수 기본 출력
printf("%010d", 123);	0 0 0 0 0 0 0 1 2 3	빈칸 대신에 0을 출력
printf("% d", 123);	␣ 1 2 3	양수이면 빈칸을 앞에 출력
printf("%+d", 123);	+ 1 2 3	양수이면 +를 붙인다.
printf("%d", -123);	- 1 2 3	음수 기본 출력
printf("%#x", 0x10);	0 x 1 0	앞에 0x을 출력한다.

필드폭과 정밀도



필드폭과 정밀도

출력 문장	출력 결과	설명
printf("%10d", 123);	␣␣␣␣␣␣␣123	폭은 10, 우측정렬
printf("%-10d", 123);	123␣␣␣␣␣␣␣	폭은 10, 좌측정렬
printf("%10d", -123);	␣␣␣␣␣␣-123	폭은 10, 우측정렬
printf("%-10d", -123);	-123␣␣␣␣␣␣	폭은 10, 좌측정렬
printf("%10s", "abc");	␣␣␣␣␣␣␣abc	폭은 10, 우측정렬
printf("%-10s", "abc");	abc␣␣␣␣␣␣␣	폭은 10, 좌측정렬

필드폭과 정밀도

출력 문장	출력 결과	설명
printf("%f", 1.23456789);	1.234568	소수점 이하 6자리
printf("%10.3f", 1.23456789);	1.235	소수점 이하 3자리
printf("%-10.3f", 1.23456789);	1.235	좌측 정렬
printf("%.3f", 1.23456789);	1.235	소수점 이하 3자리

형식 지정자

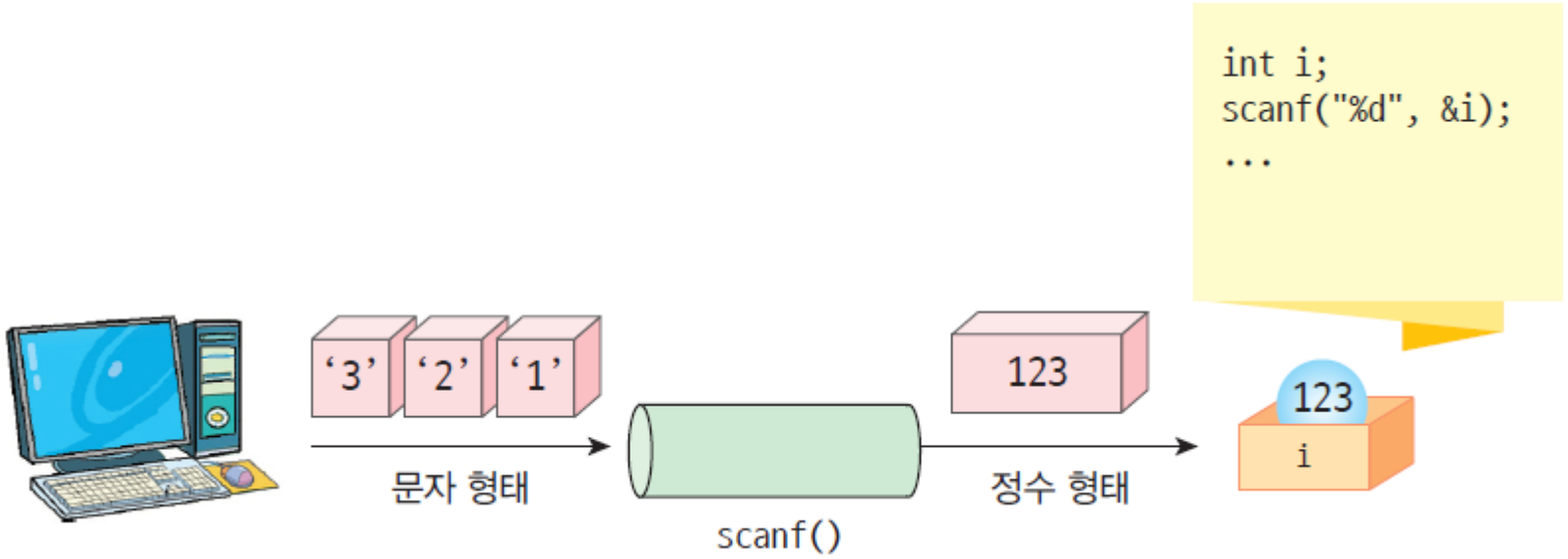
형식 지정자	설명	출력 예
%d	부호있는 10진수 형식으로 출력	255
%i	부호있는 10진수 형식으로 출력	255
%u	부호없는 10진수 형식으로 출력	255
%o	부호없는 8진수 형식으로 출력	377
%x	부호없는 16진수 형식으로 출력, 소문자로 표기	fe
%X	부호없는 16진수 형식으로 출력, 대문자로 표기	FE
%f	소수점 고정 표기 형식으로 출력	123.456
%e	지수 표기 형식으로 출력, 지수 부분을 e로 표시	1.23456e+2
%E	지수 표기 형식으로 출력, 지수 부분을 E로 표시	1.23456E+2
%g	%e형식과 %f 형식 중 더 짧은 형식으로 출력	123.456
%G	%E형식과 %f 형식 중 더 짧은 형식으로 출력	123.456
%p	포인터 형식으로 출력	0027FDD0

중간 점검

1. printf()에서 변수나 수식의 값을 출력하는 형식을 지정하는 문자열은 _____이다.
2. printf()에서 정렬(alignment)을 구체적으로 지시하지 않으면 기본적으로 _____정렬된다.
3. 실수를 지수 표기법으로 출력하는데 사용되는 형식 지정자는 _____이다.
4. 정수를 필드폭 6으로 출력하려면 %___d로 하여야 한다.
5. 실수를 필드폭 10이고 소수점 이하 자리수를 6자리로 출력하려면 %_____f로 하여야 한다.
6. 출력값을 왼쪽 정렬시키는 플래그는 ___이다.
7. 실수 출력의 경우, 정밀도를 지정하지 않았을 경우, 소수점 이하 자리수는 기본적으로 _____개가 된다.

scanf()를 이용한 입력

- 문자열 형태의 입력을 사용자가 원하는 형식으로 변환한다.



예제

```
#include <stdio.h>

int main(void)
{
    int a, b;

    printf("6개의 숫자로 이루어진 정수를 입력하시오:");
    scanf("%3d%3d", &a, &b);
    printf("입력된 정수는 %d, %d\n", a, b);

    return 0;
}
```

```
6개의 숫자로 이루어진 정수를 입력하시오:123456
입력된 정수는 123, 456
```

예제

```
#include <stdio.h>

int main(void)
{
    int a, b;

    printf("6개의 숫자로 이루어진 정수를 입력하시오:");
    scanf("%3d%3d", &a, &b);
    printf("입력된 정수는 %d, %d\n", a, b);

    return 0;
}
```

```
6개의 숫자로 이루어진 정수를 입력하시오:123456
입력된 정수는 123, 456
```

필드로 지정하여서 읽기

```
#include <stdio.h>
int main(void)
{
        int a, b;

        printf("5개의  숫자로 이루어진 정수를 입력하시오: ");
        scanf("%3d%3d", &a, &b);
        printf("입력된 정수는 %d, %d\n", a, b);

        return 0;
}
```

3글자씩 나우어서 읽는다.

```
5개의 숫자로 이루어진 정수를 입력하시오: 123456
입력된 정수는 123, 456
```

8진수, 16진수 입력

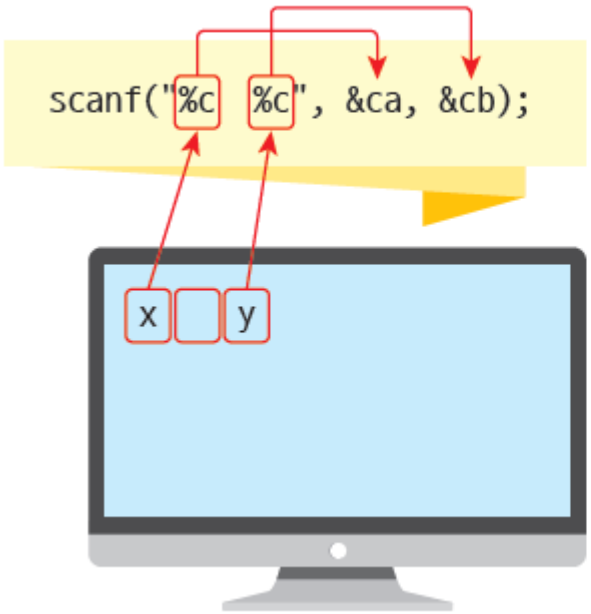
```
#include <stdio.h>

int main(void)
{
        int d, o, x;

        scanf("%d %o %x", &d, &o, &x);
        printf("d=%d o=%d x=%d\n", d, o, x);

return 0;
}
```

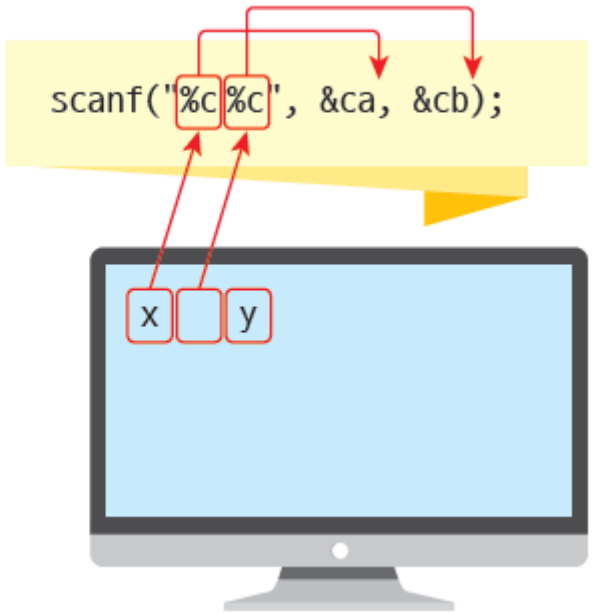
```
10
10
10
d=10 o=8 x=16
```

문자와 문자열 입력

분류	형식 지정자	설명
문자형	%c	char형으로 입력받음
	%s	공백 문자가 아닌 문자부터 공백 문자가 나올 때까지를 문자열로 변환하여 입력받음.
	%[abc]	대괄호 안에 있는 문자 a,b,c로만 이루어진 문자열을 읽어 들인다.
	%[^abc]	대괄호 안에 있는 문자 a,b,c만을 제외하고 다른 문자들로 이루어진 문자열을 읽어 들인다.
	%[0-9]	0에서 9까지의 범위에 있는 문자들로 이루어진 문자열을 읽어 들인다.

문자와 문자열 입기





예제

```c
#include <stdio.h>

int main(void)
{
	char c;
	char s[80], t[80];

	printf("스페이스로 분리된 문자열을 입력하시오:");
	scanf("%s%c%s", s, &c, t);

	printf("입력된 첫번째 문자열=%s\n", s);
	printf("입력된 문자=%c\n", c);
	printf("입력된 두번째 문자열=%s\n", t);

	return 0;
}
```

```
스페이스로 분리된 문자열을 입력하시오:Hello World
입력된 첫번째 문자열=Hello
입력된 문자=
입력된 두번째 문자열=World
```

문자집합으로 입기

```
#include <stdio.h>

int main(void)
{
	char s[80];

	printf("문자열을 입력하시오:");
	scanf("%[abc]", s);

	printf("입력된 문자열=%s\n", s);

	return 0;
}
```

```
문자열을 입력하시오:abcdef
입력된 문자열=abc
```

반환값 이용

```
#include <stdio.h>
int main(void)
{
        int x, y, z;
        if (scanf("%d%d%d", &x, &y, &z) == 3)
                printf("정수들의 합은 %d\n", x+y+z);
        else
                printf("입력값이 올바르지 않습니다.\n");
        return 0;
}
```

```
10 20 30
정수들의 합은 60
```

```
a b c
입력이 올바르지 않음
```

scanf() 사용시 주의점

- 입력값을 저장할 변수의 주소를 전달
  - int i;
  - scanf("%d", i); // 오류!!
- 배열의 이름은 배열을 가리키는 포인터
  - int str[80];
  - scanf("%s", str); // 올바름
  - scanf("%s", &str); // 오류!!

scanf() 사용시 주의점

- 충분한 공간을 확보
  - int str[80];
  - scanf("%s", str); // 입력된 문자의 개수가 79를 초과하면 치명적인 오류 발생

- scanf()의 형식 제어 문자열의 끝에 줄바꿈 문자 '\n'을 사용하는 것은 해당 문자가 반드시 입력되어야 한다는 의미
  - scanf("%d\n", &i);// 잘못됨!!

중간 점검

1. scanf()에서 double 값을 입력받을 때 사용하는 형식 지정자는 _____이다.
2. 여러 개의 입력을 받는 경우, scanf()는 _____을 이용하여 각각의 입력을 분리한다

파일이 필요한 이유

```
int main(void)
{
   ...
   ...
   ...
}
```

프로그램

메모리

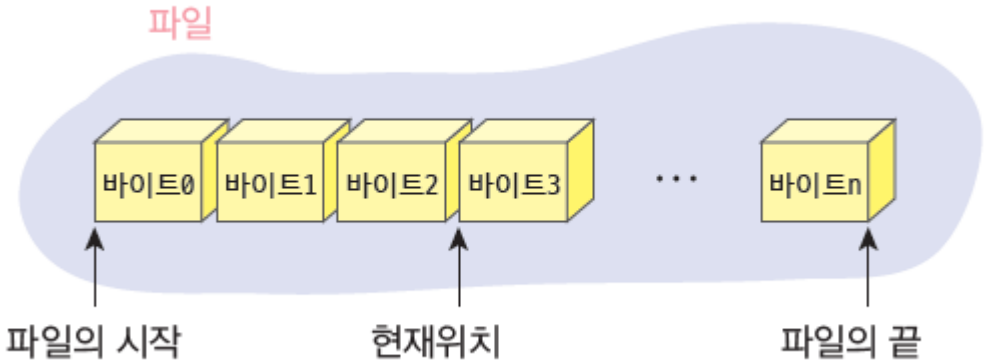
변수, 배열, 구조체 등은 모두 메모리에 만들이지고 이것들은 모두 전원이 꺼지면 사라진다.

하드 디스크

하드 디스크에 파일 형태로 저장하면 전원이 꺼지더라도 데이터가 보존된다.

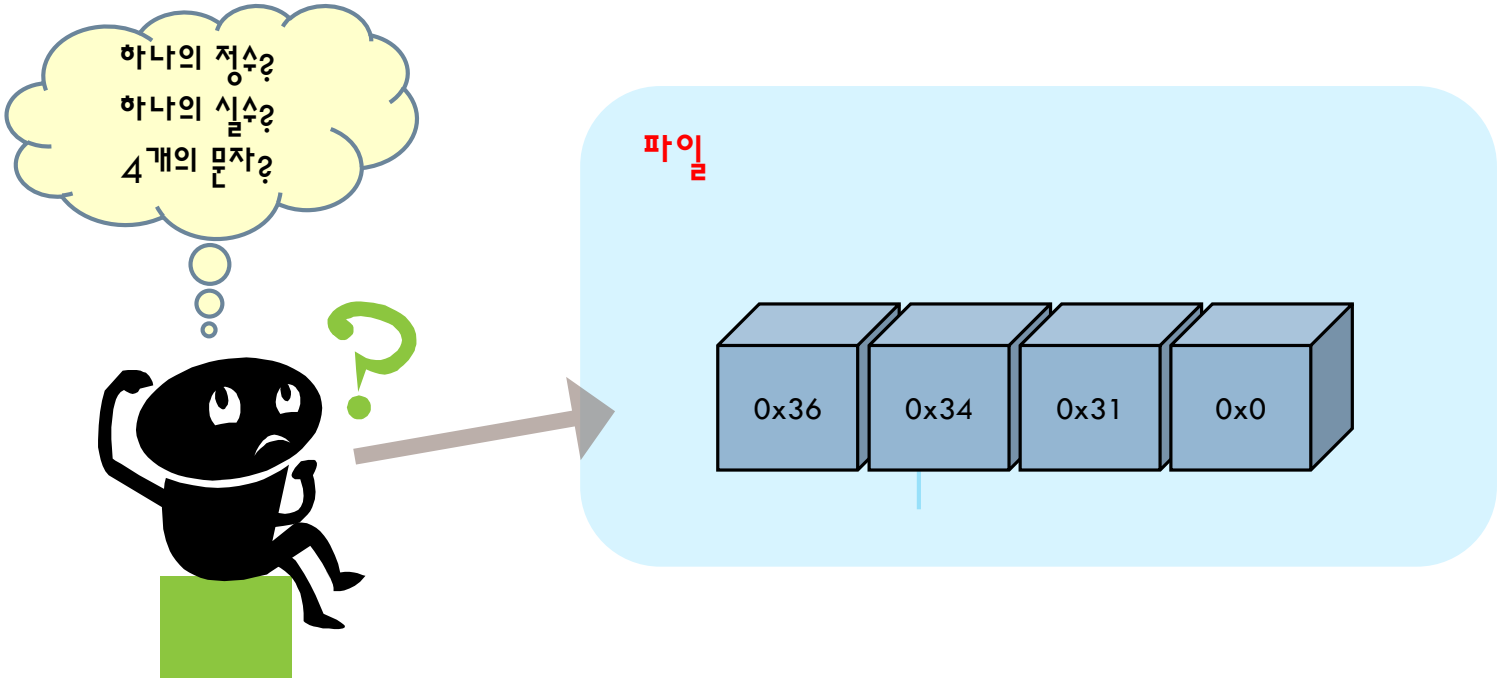
파일의 개념

- C에서의 파일은 일련의 연속된 바이트
- 모든 파일 데이터들은 결국은 바이트로 바뀌어서 파일에 저장
- 이들 바이트들을 어떻게 해석하느냐는 전적으로 프로그래머의 책임



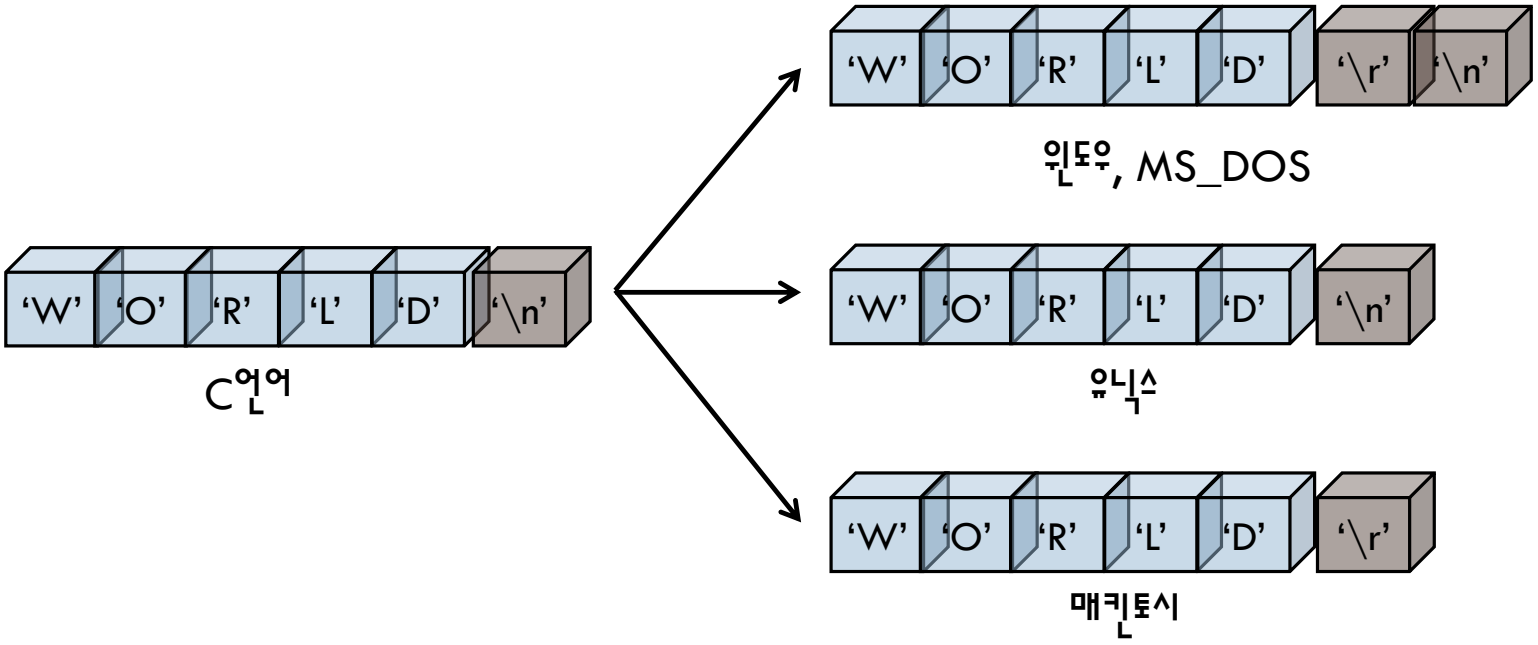
파일

- 파일에 4개의 바이트가 들어 있을 때 이것을 int형의 정수 데이터로도 해석할 수 있고 아니면 float형 실수 데이터로도 해석할 수 있다



텍스트 파일(text file)

- 텍스트 파일은 사람이 읽을 수 있는 텍스트가 들어 있는 파일
  - (예) C 프로그램 소스 파일이나 메모장 파일
- 텍스트 파일은 아스키 코드를 이용하여 저장
- 텍스트 파일은 연속적인 라인들로 구성



이진 파일(binary file)

- 이진 파일은 사람이 읽을 수는 없으나 컴퓨터는 읽을 수 있는 파일
- 이진 데이터가 직접 저장되어 있는 파일
- 이진 파일은 텍스트 파일과는 달리 라인들로 분리되지 않는다.
- 모든 데이터들은 문자열로 변환되지 않고 입출력
- 이진 파일은 특정 프로그램에 의해서만 판독이 가능
  - (예) C 프로그램 실행 파일, 사운드 파일, 이미지 파일

파일 처리의 개요

- 파일을 다룰 때는 반드시 다음과 같은 순서를 지켜야 한다.

- 디스크 파일은 FILE 구조체를 이용하여 접근
- FILE 구조체를 가리키는 포인터를 파일 포인터(file pointer)

파일 열기

Syntax: 파일열기

예

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

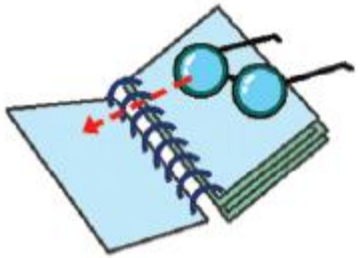
파일 이름

파일 모드

파일 모드

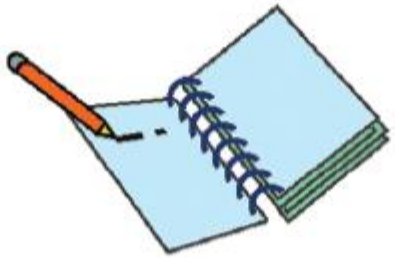
모드	설명
"r"	읽기 모드로 파일을 연다. 만약 파일이 존재하지 않으면 오류가 발생한다.
"w"	쓰기 모드로 새로운 파일을 생성한다. 파일이 이미 존재하면 기존의 내용이 지워진다.
"a"	추가 모드로 파일을 연다. 만약 기존의 파일이 있으면 데이터가 파일의 끝에 추가된다. 파일이 없으면 새로운 파일을 만든다.
"r+"	읽기 모드로 파일을 연다. 쓰기 모드로 전환할 수 있다. 파일이 반드시 존재하여야 한다.
"w+"	쓰기 모드로 새로운 파일을 생성한다. 읽기 모드로 전환할 수 있다. 파일이 이미 존재하면 기존의 내용이 지워진다.
"a+"	추가 모드로 파일을 연다. 읽기 모드로 전환할 수 있다. 데이터를 추가하면 EOF 마커를 추가된 데이터의 뒤로 이동한다. 파일이 없으면 새로운 파일을 만든다.
"t"	텍스트 파일 모드로 파일을 연다.
"b"	이진 파일 모드로 파일을 연다.

기본적인 파일 모드



"r"

파일의 처음 부터 읽는다.



"w"

파일의 처음 부터 쓴다.
만약 파일이 존재하면 기존의 내용이 지워진다.

"a"

파일의 끝에 쓴다.
파일이 없으면 생성 된다.

주의할 점

- 기본적인 파일 모드에 "t"나 "b"를 붙일 수 있다.

- "a" 나 "a+" 모드는 **추가 모드(append mode)**라고 한다. 추가 모드로 파일이 열리면, 모든 쓰기 동작은 파일의 끝에서 일어난다. 따라서 파일 안에 있었던 기존의 데이터는 절대 지워지지 않는다.

- "r+", "w+", "a+" 파일 모드가 지정되면 읽고 쓰기가 모두 가능하다. 이러한 모드를 **수정 모드(update mode)**라고 한다. 읽기 모드에서 쓰기 모드로, 또는 쓰기 모드에서 읽기 모드로 전환하려면 반드시 fflush(), fsetpos(), fseek(), rewind() 중의 하나를 호출하여야 한다.

예제

```c
#include <stdio.h>
int main(void)
{
    FILE *fp = NULL;

    fp = fopen("sample.txt", "w");

    if( fp == NULL )
        printf("파일 열기 실패\n");
    else
        printf("파일 열기 성공\n");

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


```
파일 열기 성공
```

파일 닫기와 삭제

Syntax: 파일닫기

예 `fclose(fp);`

FILE 포인터

중간 점검

1. 파일은 일련의 연속된 _____라고 생각할 수 있다.
2. 파일에는 사람이 읽을 수 있는 텍스트가 들어 있는 _____파일과 사람은 읽을 수 없으나 컴퓨터는 읽을 수 있는 _____파일이 있다.
3. 파일을 여는 라이브러리 함수는 _____이다.
4. fopen()은 _____을 가리키는 포인터를 반환한다.

파일 입출력 함수

종류	입력 함수	출력 함수
문자 단위	int fgetc(FILE *fp)	int fputc(int c, FILE *fp)
문자열 단위	char *fgets(char *buf, int n, FILE *fp)	int fputs(const char *buf, FILE *fp)
서식화된 입출력	int fscanf(FILE *fp, ...)	int fprintf(FILE *fp,...)
이진 데이터	size_t fread(char *buffer, int size, int count, FILE *fp)	size_t fwrite(char *buffer, int size, int count, FILE *fp)

문자 단위 입출력

```
#include <stdio.h>
int main(void)
{
        FILE *fp = NULL;

        fp = fopen("sample.txt", "w");
        if( fp == NULL )
                printf("파일 열기 실패\n");
        else
                printf("파일 열기 성공\n");

        fputc('a', fp);
        fputc('b', fp);
        fputc('c', fp);
        fclose(fp);
        return 0;
}
```

sample.txt

```
abc
```

파일 열기 성공

문자 단위 입출력

```c
#include <stdio.h>
int main(void)
{
	FILE *fp = NULL;
	int c;
	fp = fopen("sample.txt", "r");
	 if( fp == NULL )
		printf("파일 열기 실패\n");
	else
		printf("파일 열기 성공\n");

	while((c = fgetc(fp)) != EOF )
		putchar(c);
	fclose(fp);
	 return 0;
}
```

sample.txt

```
abc
```

파일 열기 성공
abc

문자열 단위 입출력

Syntax: 문자열 단위 입출력

예

```
char *fgets( char *s, int n, FILE *fp );
int fputs( char *s, FILE *fp );
```

여기에 문자열을 저장

최대 개수

문자열 단위 입출력

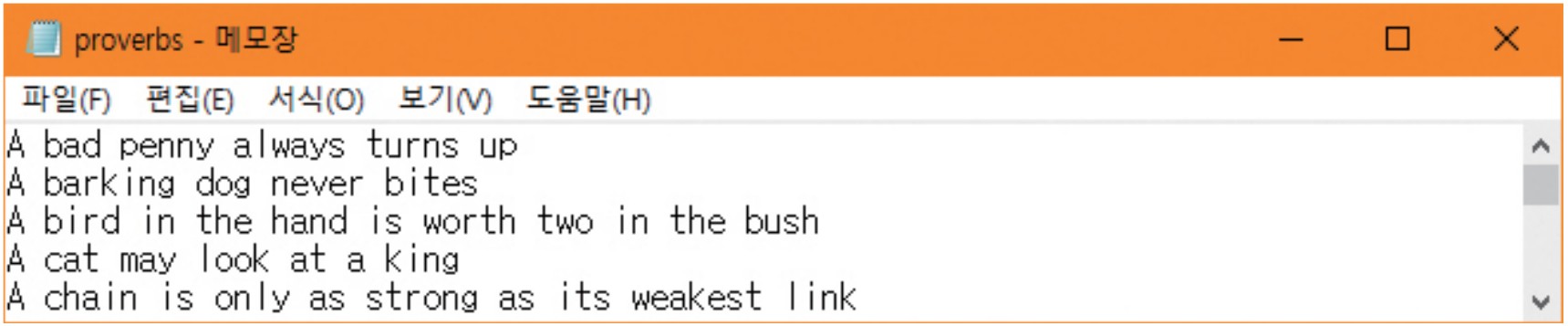
```
#include <stdio.h>
#include <stdlib.h>

int main(void)
{
    FILE *fp1, *fp2;
    char file1[100], file2[100];
    char buffer[100];

    printf("원본 파일 이름: ");
    scanf("%s", file1);

    printf("복사 파일 이름: ");
    scanf("%s", file2);

    // 첫번째 파일을 읽기 모드로 연다.
    if( (fp1 = fopen(file1, "r")) == NULL )
    {
        fprintf(stderr,"원본 파일 %s을 열 수 없습니다.\n", file1);
        exit(1);
    }
```

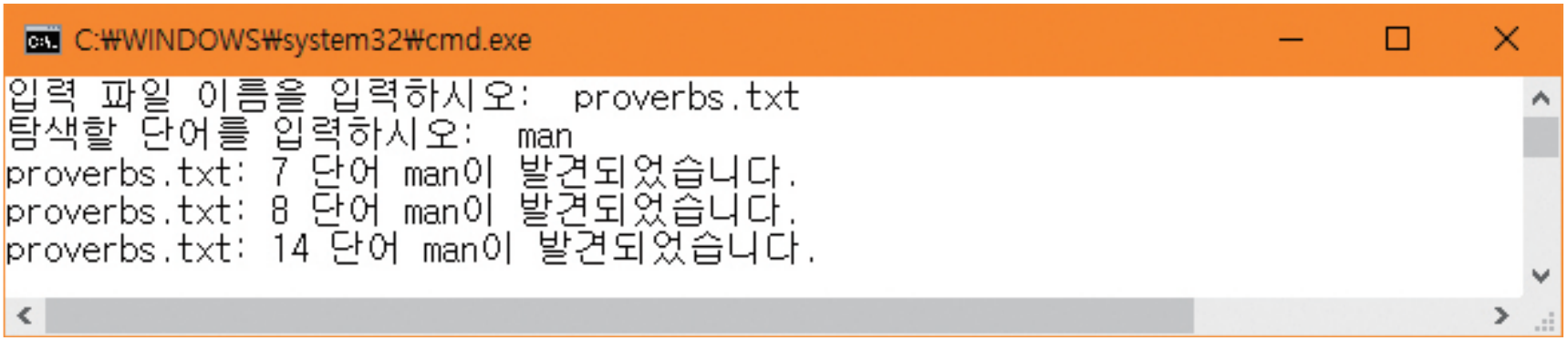
문자열 단위 입출력

```
    // 두번째 파일을 쓰기 모드로 연다.
    if( (fp2 = fopen(file2, "w")) == NULL )
    {
        fprintf(stderr,"복사 파일 %s을 열 수 없습니다.\n", file2);
        exit(1);
    }

    // 첫번째 파일을 두번째 파일로 복사한다.
    while( fgets(buffer, 100, fp1) != NULL )
        fputs(buffer, fp2);

    fclose(fp1);
    fclose(fp2);

    return 0;
}
```

원본 파일 이름: a.txt
복사 파일 이름: b.txt

lab: 텍스트 파일에서 특정 문자열을 탐색하는 프로그램을 작성하여 보자.





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

int main(void)
{
    FILE *fp;
    char fname[128];
    char buffer[256];
    char word[256];
    int line_num = 0;

    printf("입력 파일 이름을 입력하시오: ");
    scanf("%s", fname);

    printf("탐색할 단어를 입력하시오: ");
    scanf("%s", word);
```

```
    // 파일을 읽기 모드로 연다.
    if( (fp = fopen(fname, "r")) == NULL )
    {
            fprintf(stderr,"파일 %s을 열 수 없습니다.\n", fname);
            exit(1);
    }

    while( fgets(buffer, 256, fp) ) {
            line_num++;
            if( strstr(buffer, word) ) {
               printf("%s: %d 단어 %s이 발견되었습니다.\n", fname, line_num, word );
            }
    }
    fclose(fp);

    return 0;
}
```

형식화된 입출력

Syntax: 형식화된 입출력

예

```
int fprintf( FILE *fp,  const char *format, ...);
int fscanf( FILE *fp,  const char *format, ...);
```

예제

```
int main(void)
{
    FILE *fp;
    char fname[100];
    int number, count = 0;
    char name[20];
    float score, total = 0.0;

    printf("성적 파일 이름을 입력하시오: ");
    scanf("%s", fname);

    // 성적 파일을 쓰기 모드로 연다.
    if( (fp = fopen(fname, "w")) == NULL )
    {
        fprintf(stderr,"성적 파일 %s을 열 수 없습니다.\n", fname);
        exit(1);
    }
```

예제

```
// 사용자로부터 학번, 이름, 성적을 입력받아서 파일에 저장한다.
while( 1 )
{
        printf("학번, 이름, 성적을 입력하시요: (음수이면 종료)");
        scanf("%d", &number);
        if( number < 0 ) break
        scanf("%s %f", name, &score);
        fprintf(fp, " %d %s %f", number, name, score);
}
fclose(fp);
// 성적 파일을 읽기 모드로 연다.
if( (fp = fopen(fname, "r")) == NULL )
{
        fprintf(stderr,"성적 파일 %s을 열 수 없습니다.\n", fname);
        exit(1);
}
```

예제

```
        // 파일에서 성적을 읽어서 평균을 구한다.
    while( !feof( fp ) )
    {
        fscanf(fp, "%d %s %f", &number, name, &score);
        total += score;
        count++;
    }
    printf("평균 = %f\n", total/count);
    fclose(fp);
    return 0;
}
```

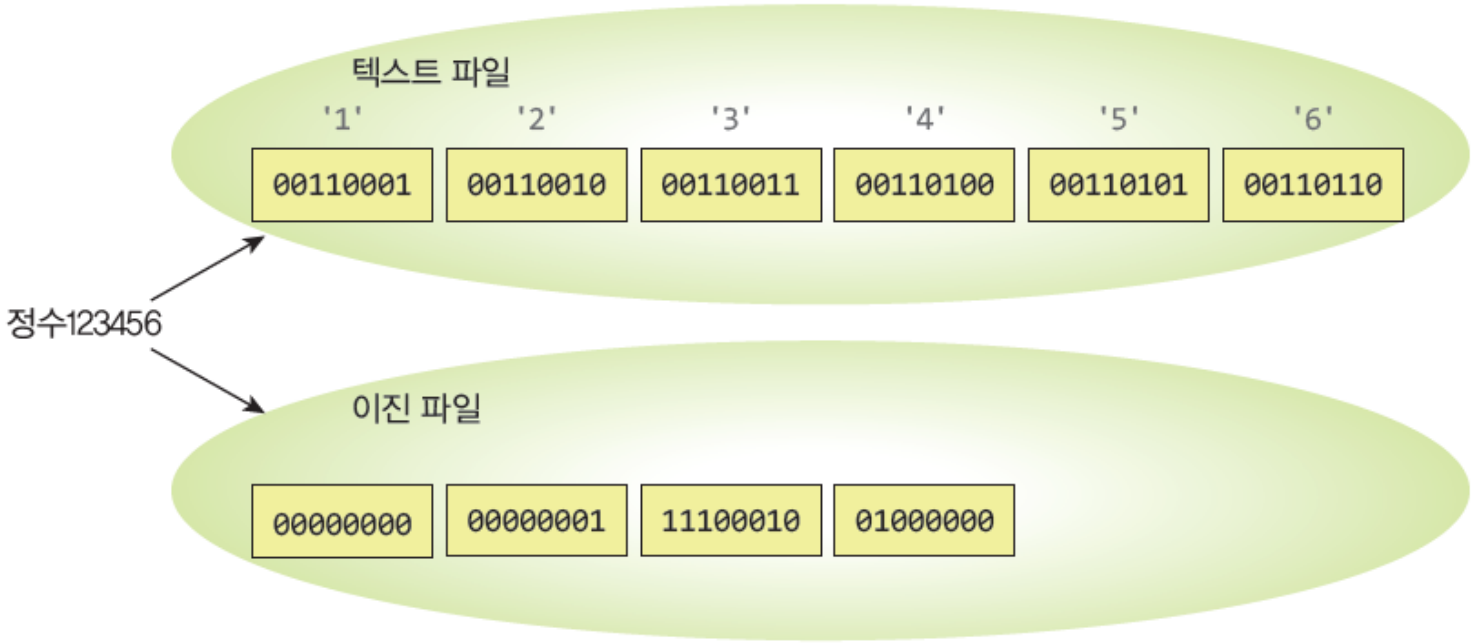
```
성적 파일 이름을 입력하시오: scores.txt
학번, 이름, 성적을 입력하시오: (음수이면 종료)1 KIM 10.0
학번, 이름, 성적을 입력하시오: (음수이면 종료)2 PARK 20.0
학번, 이름, 성적을 입력하시오: (음수이면 종료)3 LEE 30.0
학번, 이름, 성적을 입력하시오: (음수이면 종료)-1
평균 = 20.000000
```

중간 점검

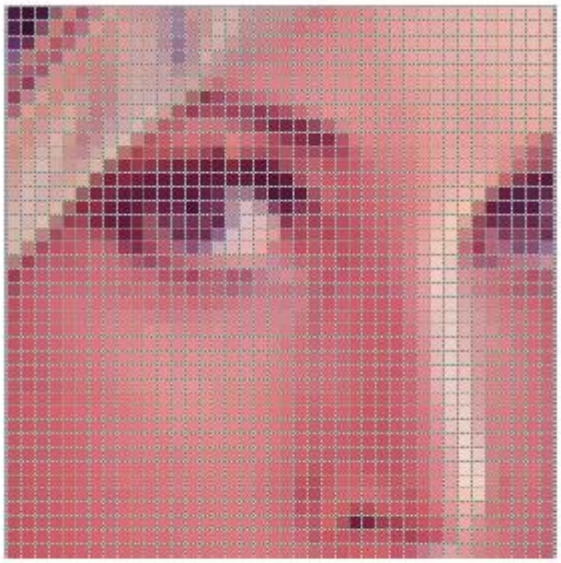
1. fgetc()의 반환형은 _____형이다.
2. 파일에서 하나의 라인을 읽어서 반환하는 함수는 _____이다.
3. 텍스트 파일에 실수나 정수를 문자열로 변경하여 저장할 때 사용하는 함수는 _____이다.
4. 텍스트 파일에서 실수나 정수를 읽는 함수는 _____이다.

이진 파일 쓰기와 읽기

- 텍스트 파일과 이진 파일의 차이점
  - *텍스트 파일*: 모든 데이터가 아스키 코드로 변환되어서 저장됨
  - *이진 파일*: 컴퓨터에서 데이터를 표현하는 방식 그대로 저장



이진 파일의 예



각 픽셀의 밝기를 2진수로 나타낸다.

이진 파일 쓰기

```
#include <stdio.h>

#define SIZE 5
int main(void)
{

	int buffer[SIZE] = { 10, 20, 30, 40, 50 };
	FILE *fp = NULL;

	fp = fopen("binary.bin", "wb");		// ①
	if( fp == NULL )
	{
		fprintf(stderr, "binary.bin 파일을열수없습니다.");
		return 1;
	}

	fwrite(buffer, sizeof(int), SIZE, fp);	// ②

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

이진 파일 모드

파일 모드	설명
"rb"	읽기 모드 + 이진 파일 모드
"wb"	쓰기 모드 + 이진 파일 모드
"ab"	추가 모드 + 이진 파일 모드
"rb+"	읽고 쓰기 모드 + 이진 파일 모드
"wb+"	쓰고 읽기 모드 + 이진 파일 모드

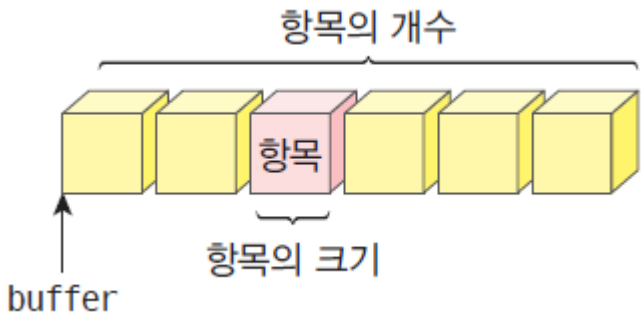
이진 파일 쓰기

Syntax: fwrite()

예

```
fwrite(buffer, sizeof(int), SIZE, fp);
```

- buffer: 메모리 블록의 주소
- sizeof(int): 항목의 크기
- SIZE: 항목의 개수
- fp: FILE 포인터



이진 파일 읽기

```
#include <stdio.h>
#define SIZE 5

int main(void)
{
	int i;
	int buffer[SIZE];
	FILE *fp = NULL;

	fp = fopen("binary.bin", "rb");
	if( fp == NULL )
	{
		fprintf(stderr, "binary.bin 파일을 열 수 없습니다.");
		return 1;
	}
	fread(buffer, sizeof(int), SIZE, fp);

	for(i=0 ; i<SIZE ; i++)
		printf("%d ", buffer[i]);

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

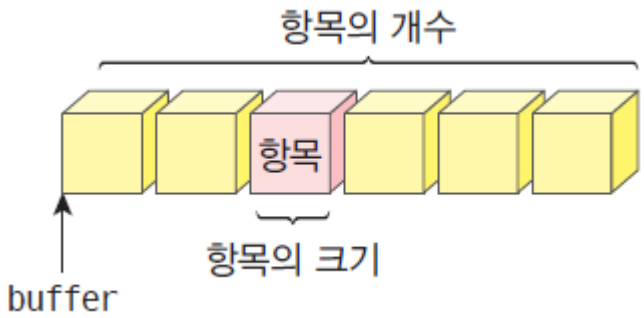
이진 파일 쓰기

Syntax: fread()

예

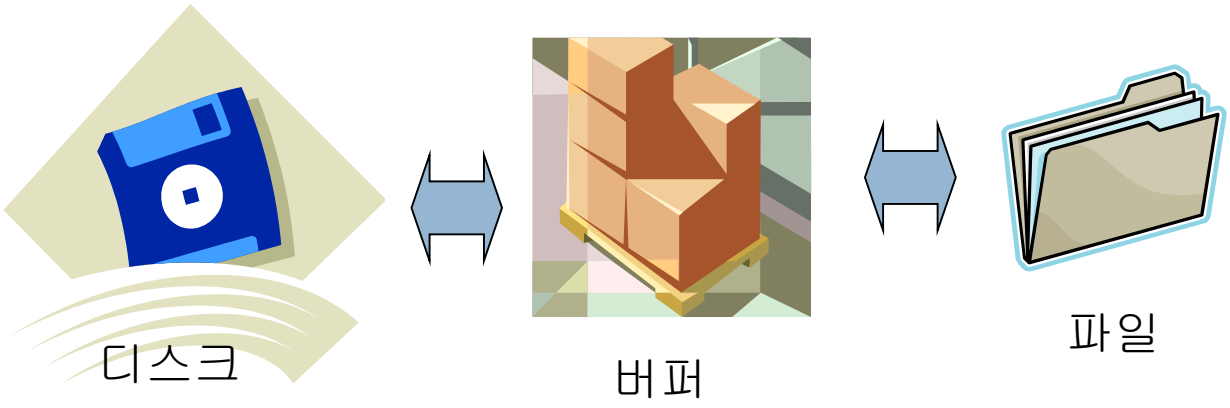
```
fread(buffer, sizeof(int), SIZE, fp);
```

- buffer: 메모리 블록의 주소
- sizeof(int): 항목의 크기
- SIZE: 항목의 개수
- fp: FILE 포인터



버퍼링

- 버퍼는 파일로부터 읽고 쓰는 데이터의 임시 저장 장소로 이용되는 메모리의 블록
- 디스크 드라이브는 블록 단위 장치이기 때문에 블록 단위로 입출력을 해야만 가장 효율적으로 동작
- 1024바이트의 블록이 일반적



버퍼링

- fflush(fp);
  - 버퍼의 내용이 디스크 파일에 써진다.

- setbuf(fp, NULL);
  - setbuf()는 스트림의 버퍼를 직접 지정하는 함수로서 만약 버퍼 자리에 NULL을 써주면 버퍼를 제거하겠다는 것을 의미한다.

lab: 이진 파일에 학생 정보 저장하기

- 학생들의 정보를 이진 파일에 저장하고 다시 읽어서 화면에 출력하는 프로그램을 작성하여 보자.

예제

```
#define SIZE 3
struct student {
	int number;			// 학번
	char name[20];		// 이름
	double gpa;			// 평점
};

int main(void)
{
	struct student table[SIZE] = {
		{ 1, "Kim", 3.99 },
		{ 2, "Min", 2.68 },
		{ 3, "Lee", 4.01 }
	};
	struct student s;
	FILE *fp = NULL;
	int i;
	// 이진 파일을 쓰기 모드로 연다.
	if( (fp = fopen("student.dat", "wb")) == NULL )
	{
		fprintf(stderr,"출력을 위한 파일을 열 수 없습니다.\n");
		exit(1);
	}
```

예제

```
    // 배열을 파일에 저장한다.
    fwrite(table, sizeof(struct student), SIZE, fp);
    fclose(fp);
    // 이진 파일을 읽기 모드로 연다.
    if( (fp = fopen("student.dat", "rb")) == NULL )
    {
            fprintf(stderr,"입력을 위한 파일을 열 수 없습니다.\n");
            exit(1);
    }
    for(i = 0;i < SIZE; i++)
    {
            fread(&s, sizeof(struct student), 1, fp);
            printf("학번 = %d, 이름 = %s, 평점 = %f\n", s.number, s.name, s.gpa);
    }
    fclose(fp);
    return 0;
}
```

학번 = 1, 이름 = Kim, 평점 = 3.990000
학번 = 2, 이름 = Min, 평점 = 2.680000
학번 = 3, 이름 = Lee, 평점 = 4.010000

lab: 이미지 파일 복사하기

- 여기서는 이진 파일을 복사하는 프로그램을 작성하여 보자.

예제

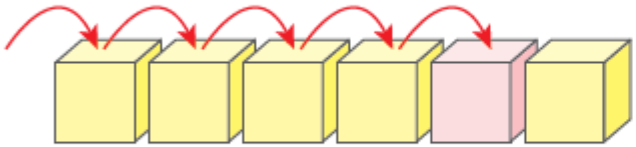
```
#include <stdio.h>

int main(void)
{
        FILE *src_file, *dst_file;
        char filename[100];
        char buffer[1024];
        int r_count;

        printf("이미지 파일 이름: ");
        scanf("%s", filename);

        src_file = fopen(filename, "rb");
        dst_file = fopen("copy.jpg", "wb");
        if( src_file==NULL || dst_file ==NULL ){
                fprintf(stderr, "파일 열기 오류 \n");
                return 1;
        }
```

예제

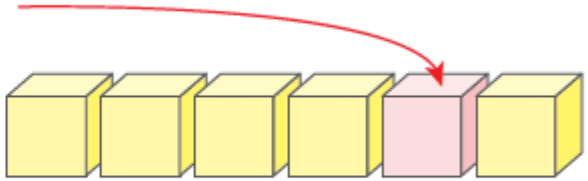
```
        while((r_count = fread(buffer, 1, sizeof(buffer), src_file)) > 0) {
                int w_count = fwrite(buffer, 1, r_count, dst_file);
                if( w_count < 0 ) {
                        fprintf(stderr, "파일 쓰기 오류 \n");
                        return 1;
                }
                if( w_count < r_count ) {
                        fprintf(stderr, "미디어 쓰기 오류 \n");
                        return 1;
                }
        }
        printf("copy.jpg로 이미지 파일이 복사됨 \n");
        fclose(src_file);
        fclose(dst_file);
        return 0;
}
```

임의 접근 파일

- **순차 접근(sequential access)** 방법: 데이터를 파일의 처음부터 순차적으로 읽거나 기록하는 방법
- **임의 접근(random access)** 방법: 파일의 어느 위치에서든지 읽기와 쓰기가 가능한 방법



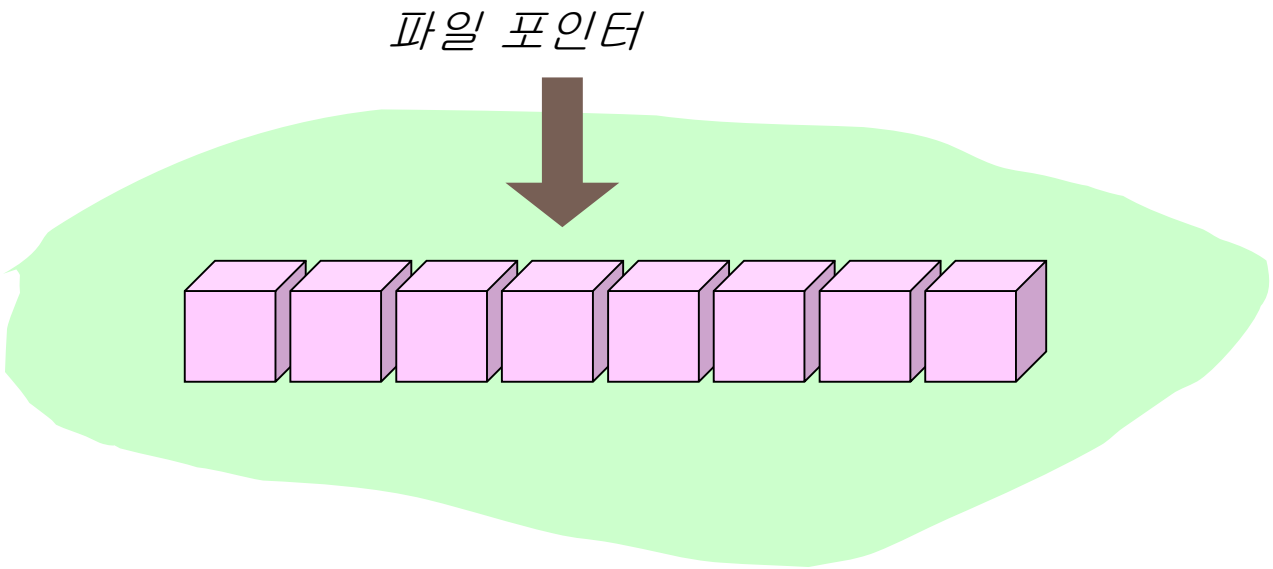
순차 접근파일



임의 접근파일

임의 접근 파일의 원리

- 파일 포인터: 읽기와 쓰기 동작이 현재 어떤 위치에서 이루어지는 지를 나타낸다.



- 강제적으로 파일 포인터를 이동시키면 임의 접근이 가능

fseek()

Syntax: fseek()

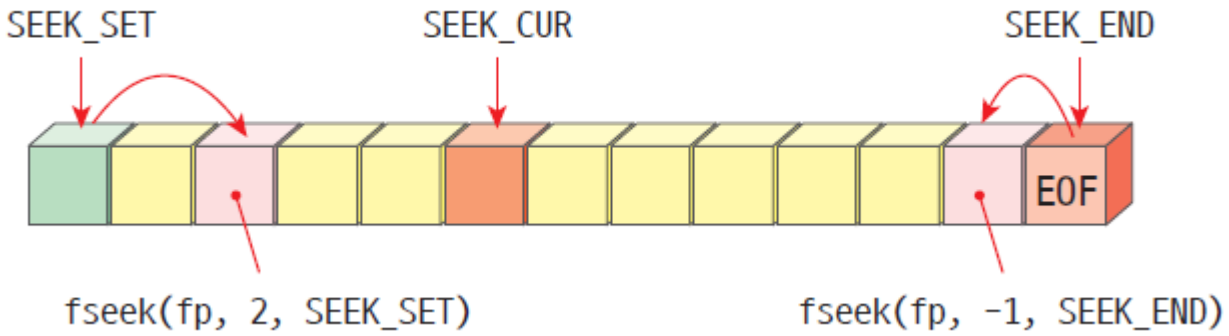
예

```
int fseek(FILE *fp, long offset, int origin);
```

FILE 포인터 — 거리 — 기준 위치

상수	값	설명
SEEK_SET	0	파일의 시작
SEEK_CUR	1	현재 위치
SEEK_END	2	파일의 끝

fseek()



ftell()

Syntax: ftell()

예

```
long ftell(FILE *fp);
```

예제

```
#include <stdio.h>

int main (void)
{
	FILE *fp;

	char buffer[100];

	fp = fopen("sample.txt", "wt");
	fputs( "ABCDEFGHIJKLMNOPQRSTUVWXYZ" , fp );
	fclose(fp);

	fp = fopen("sample.txt", "rt");


	fseek( fp , 3 , SEEK_SET );
	printf("fseek(fp, 3, SEEK_SET) = %c \n", fgetc(fp));

	fseek( fp , -1 , SEEK_END );
	printf("fseek(fp, -1, SEEK_END) = %c \n", fgetc(fp));

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

예제

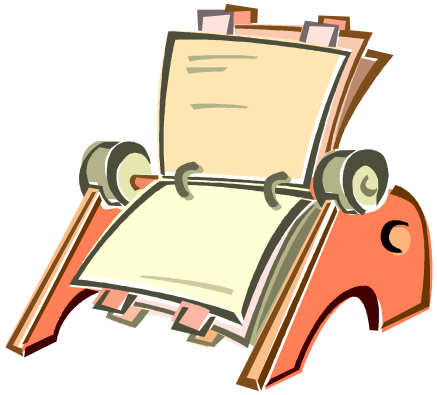
```
fp, 3, SEEK_SET) = D
fseek(fp, -1, SEEK_END) = Z
```

중간 점검

1. 파일의 처음부터 순차적으로 읽거나 쓰는 방법을 _____이라고 한다.
2. 파일의 어느 위치에서나 읽고 쓰기가 가능한 방법을 _____이라고 한다.
3. 파일에서 읽기나 쓰기가 수행되면 파일의 현재의 위치를 표시하는 _____가 갱신된다.
4. 파일의 파일 포인터를 알아내는 함수는 _____이다.

mini project: 주소록 만들기

- 자신과 친한 사람들의 정보를 저장하고 업데이트할 수 있는 간단한 프로그램을 작성하여 보자.
- 입력하거나 업데이트한 데이터는 파일로 저장된다. 저장된 데이터에 대하여 검색할 수 있다.
- 자기에게 필요한 여러 가지 사항들을 저장할 수 있도록 하자. 즉 자신만의 간단한 데이터베이스 시스템을 작성하여 보자.



실행 결과

```
====================
1. 추가
2. 수정
3. 검색
4. 종료
====================
메뉴를 선택하세요: 1
이름: 홍길동
주소: 서울시 종로구 1번지
휴대폰: 010-123-4567
특징: 싸움을 잘함, 변신에 능함
...
```

힌트

- 적합한 것은 “a+"모드이다. 주로 추가, 탐색 할 예정
- 파일에서 읽기 전에 무조건 fseek()를 해주어야 한다.
- 수정할 때는 차라리 새로운 파일을 생성하여서 거기에 전체를 다시 기록하는 것이 낫다.

예제

```
#include <stdio.h>
#include <string.h>
#define SIZE 100
typedef struct person {                    // 연락처를 구조체로 표현한다.
        char name[SIZE];                   // 이름
        char address[SIZE];  // 주소
        char mobilephone[SIZE];            // 휴대폰
        char desc[SIZE];                   // 특징
} PERSON;
void menu();
PERSON get_record();
void print_record(PERSON data);
void add_record(FILE *fp);
void search_record(FILE *fp);
void update_record(FILE *fp);
```

제

```
int main(void)
{
        FILE *fp;
        int select;
        // 이진 파일을 추가 모드로 오픈한다.
        if( (fp = fopen("address.dat", "a+")) == NULL ) {
                fprintf(stderr,"입력을 위한 파일을 열 수 없습니다);
                exit(1);
        }
        while(1)  {
                menu();                 // 메뉴를 표시한다
                printf("정수값을 입력하시오: ");          // 사용자로부터
정수를 받는다
                scanf("%d",&select);
                switch(select) {
                case 1:    add_record(fp); break;         // 데이터를 추가한다
                case 2:    update_record(fp); break;      // 데이터를 수정한다
                case 3:    search_record(fp); break;      // 데이터를 탐색한다
                case 4:    return 0;
                }
        }
        fclose(fp);             // 이진 파일을 닫는다
        return 0;
}
```

```
// 사용자로부터 데이터를 받아서 구조체로 반환한다
PERSON get_record()
{
        PERSON data;
        fflush(stdin);                  // 표준 입력의 버퍼를 비운다
        printf("이름);      gets(data.name);    // 이름을 입력받는다
        printf("주소);      gets(data.address); // 주소를 입력받는다
        printf("휴대폰);    gets(data.mobilephone);     // 휴대폰 번호를
입력받는다
        printf("특징);      gets(data.desc);    // 특징을 입력 받는다
        return data;
}
// 구조체 데이터를 화면에 출력한다.
void print_record(PERSON data)
{
        printf("이름\n", data.name);            printf("주소\n", data.address);
        printf("휴대폰\n", data.mobilephone);   printf("특징\n", data.desc);
}
```

```
// 메뉴를 화면에 표시하는 함수
void menu()
{
        printf("====================\n");
        printf(" 1. 추가\n 2. 수정\n 3. 검색\n 4. 종료\n");
        printf("====================\n");
}
// 데이터를 추가한다
void add_record(FILE *fp)
{
        PERSON data;
        data = get_record();          // 사용자로부터 데이터를 받아서 구조체에
저장
        fseek(fp, 0, SEEK_END);       // 파일의 끝으로 간다
        fwrite(&data, sizeof(data), 1, fp);       // 구조체 데이터를 파일에 쓴다
}
```

```
// 데이터를 탐색한다
void search_record(FILE *fp)
{
        char name[SIZE];
        PERSON data;
        fseek(fp, 0, SEEK_SET);         // 파일의 처음으로 간다
        fflush(stdin);
        printf("탐색하고자 하는 사람의 이름");
        gets(name);                     // 이름을 입력받는다
        while(!feof(fp)){               // 파일의 끝까지 반복한다
                fread(&data, sizeof(data), 1, fp);
        if( strcmp(data.name, name) == 0 ){     // 이름을 비교한다
                print_record(data);
                break;
        }
        }
}
// 데이터를 수정한다
void update_record(FILE *fp)
{
        //...
}
```

도전문제

- search_record()에서 탐색이 실패했으면 에러 메시지를 출력하도록 코드를 추가하여 보라.
- update_record() 함수를 구현하여 보자. 앞에서 언급한 대로 수정된 부분만 파일에 덮어쓰는 것은 상당히 어렵다. 따라서 수정된 전체 내용을 읽어서 새로운 파일에 쓰도록 해보자.

Q & A