


# 제 15장 파일 입출력

# 이번 장에서 학습할 내용

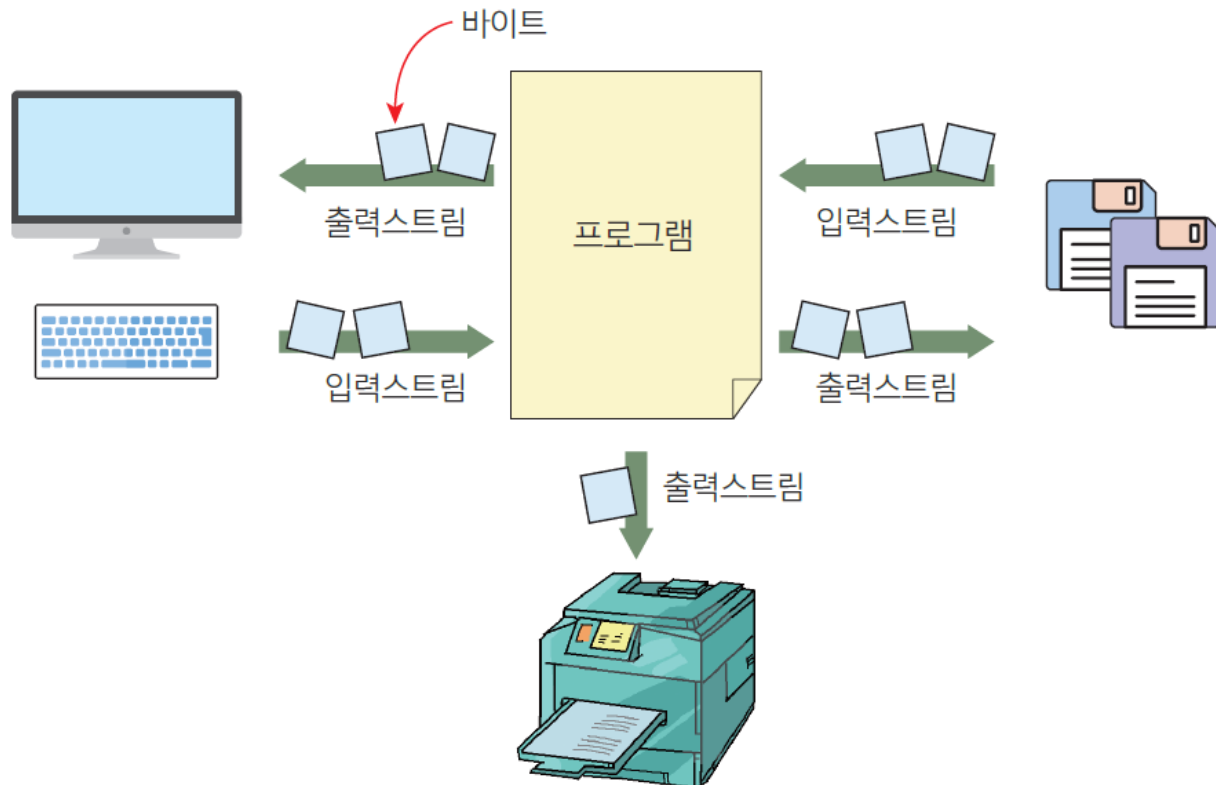
- 
- 스트립의 개념
  - 표준 입출력
  - 파일 입출력
  - 입출력 관련 함수

입출력에  
관련된  
개념들과  
함수들에  
대하여  
학습한다.



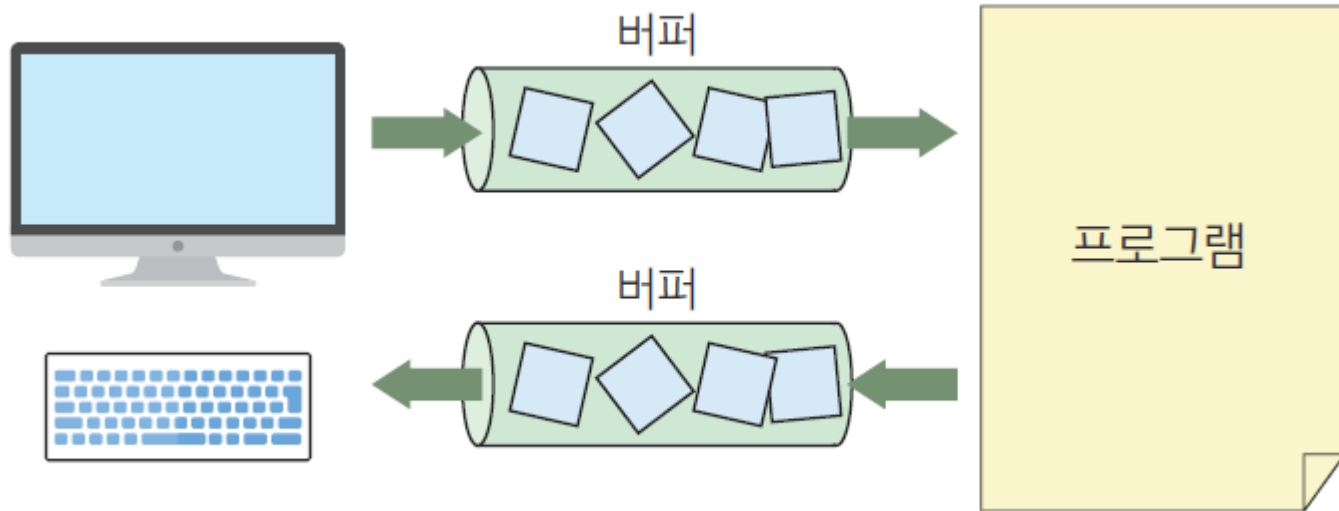
# 스트림의 개념

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



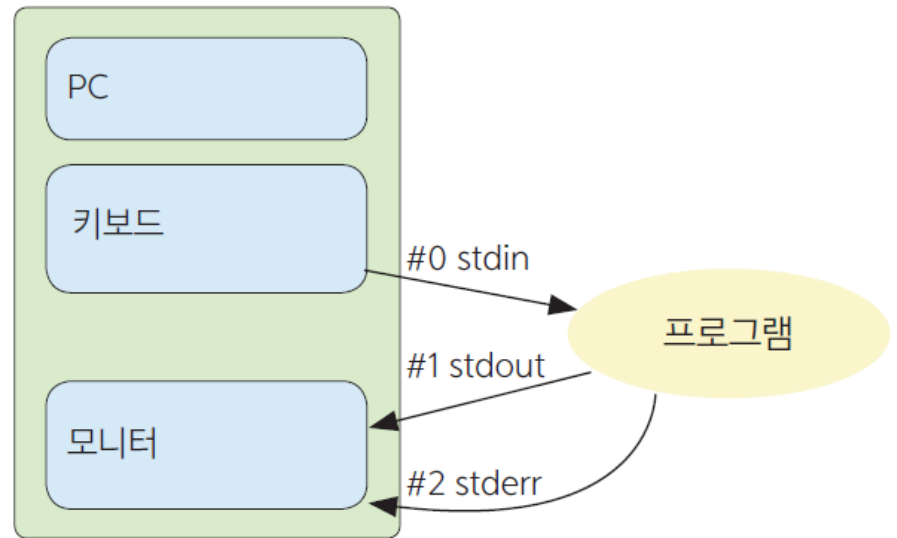
# 스트림과 버퍼

- 스트림에는 기본적으로 버퍼가 포함되어 있다.



# 표준 입출력 스트림

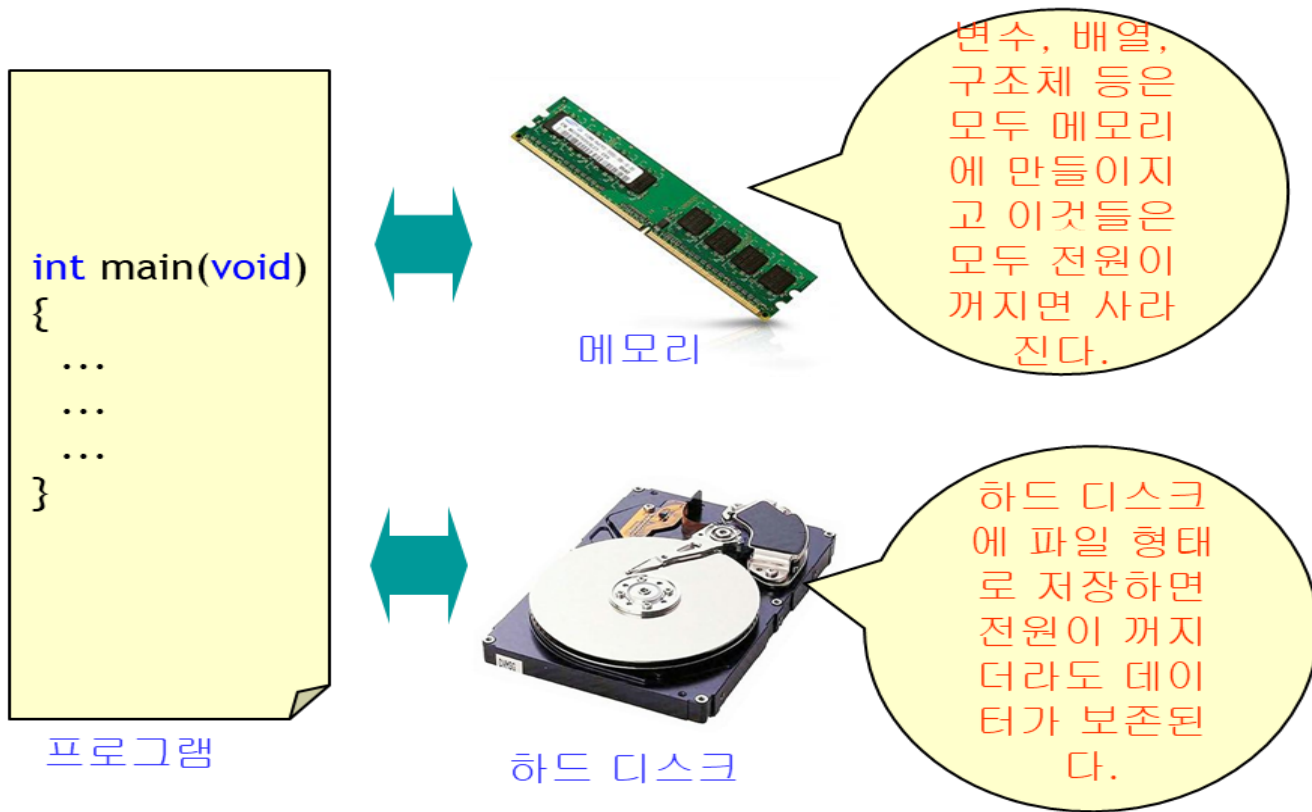
이름	스트림	연결 장치
stdin	표준 입력 스트림	키보드
stdout	표준 출력 스트림	모니터의 화면
stderr	표준 오류 스트림	모니터의 화면



# 입출력 함수의 분류

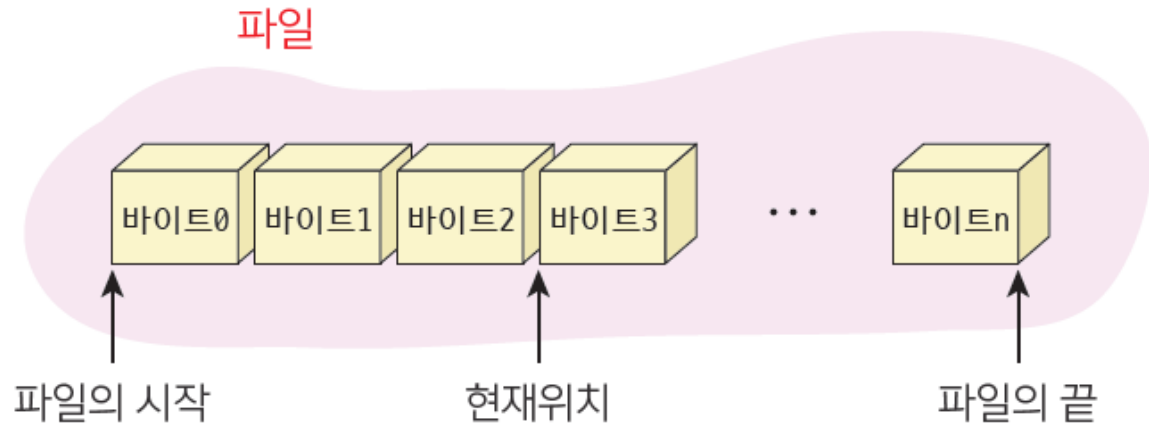
형식 \ 스트림	표준 스트림	일반 스트림	설명
형식이 없는 입출력 (문자 형태)	getchar()	fgetc(FILE *f,...)	문자 입력 함수
	putchar()	fputc(FILE *f,...)	문자 출력 함수
	gets_s()	fgets(FILE *f,...)	문자열 입력 함수
	puts()	fputs(FILE *f,...)	문자열 출력 함수
형식이 있는 입출력 (정수, 실수...)	printf()	fprintf(FILE *f,...)	형식화된 출력 함수
	scanf()	fscanf(FILE *f,...)	형식화된 입력 함수

# 파일이 필요한 이유



# 파일의 개념

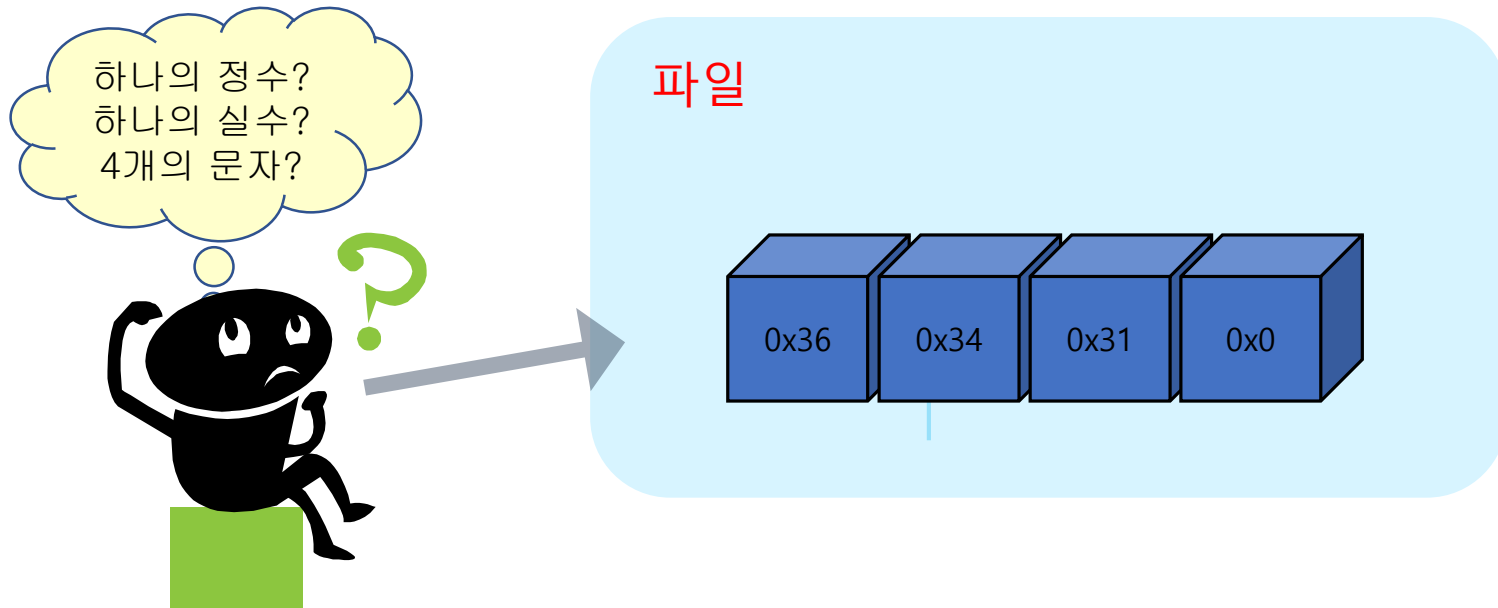
- C에서의 파일은 일련의 연속된 바이트
- 모든 파일 데이터들은 결국은 바이트로 바뀌어서 파일에 저장
- 이들 바이트들을 어떻게 해석하느냐는 전적으로 프로그래머의 책임





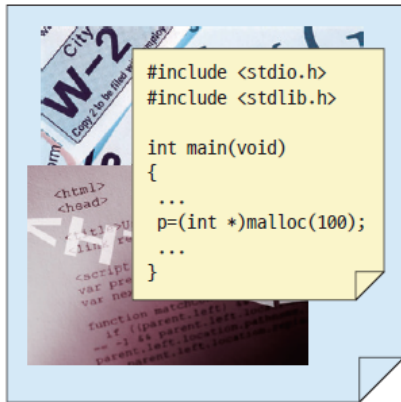
# 파일

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

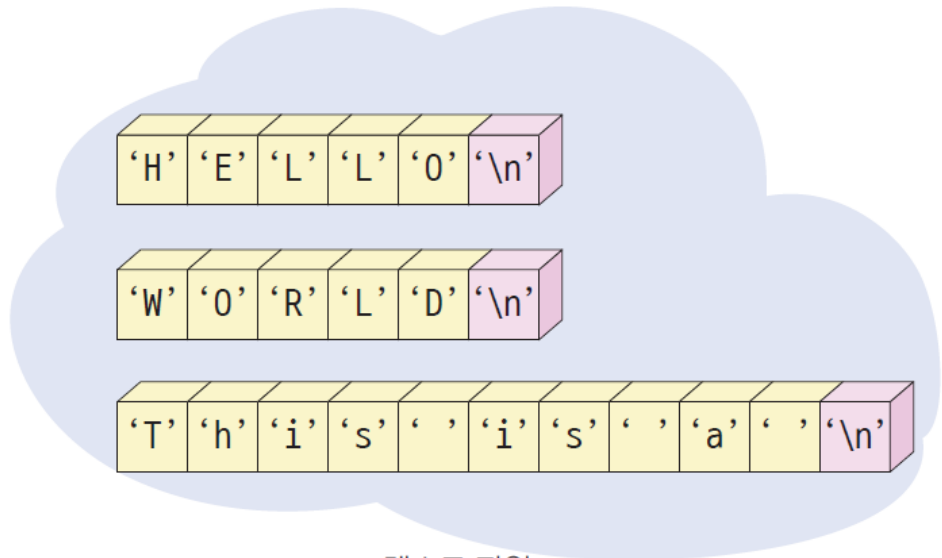


# 텍스트 파일(text file)

- 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



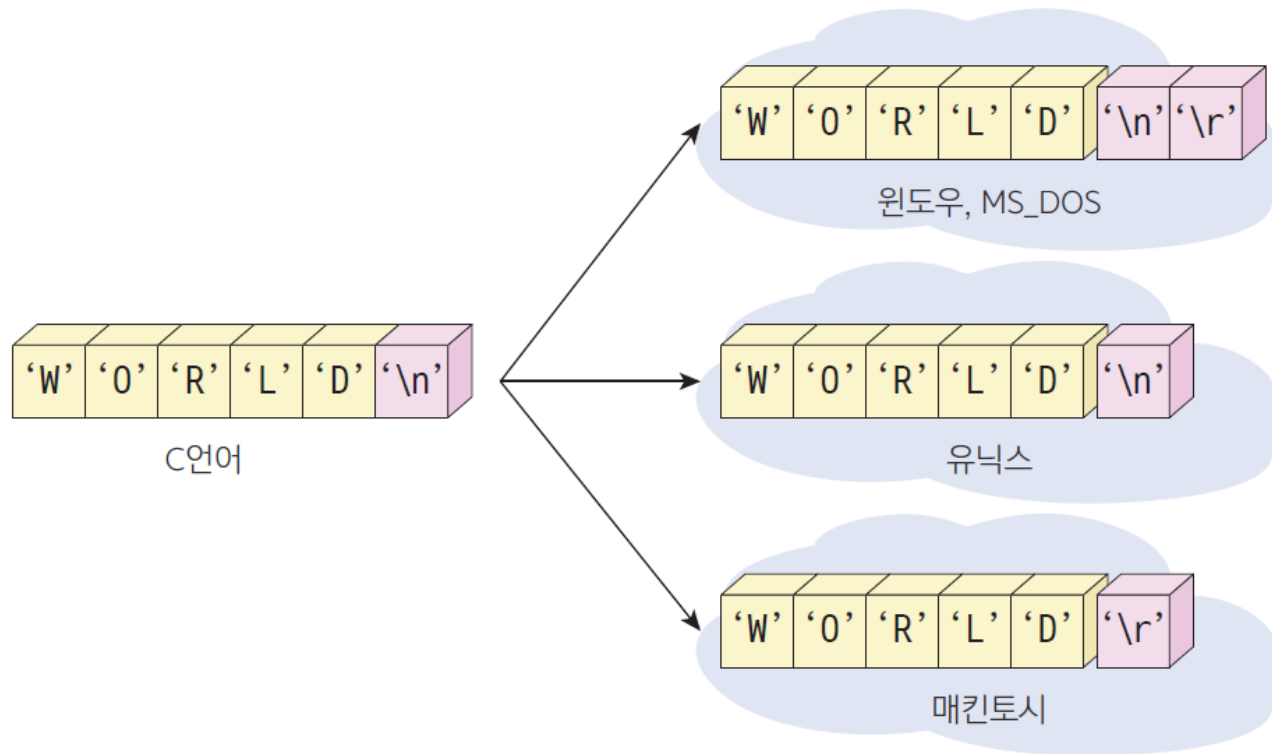
텍스트 파일: 문자로 구성된 파일



텍스트 파일

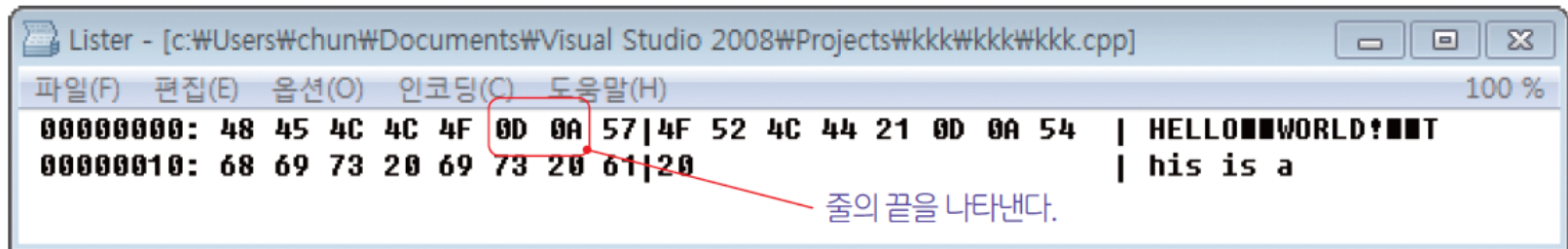
# 텍스트 파일(text file)

- 운영체제마다 줄바꿈을 표시하는 방법이 다르다.



# 윈도우에서 텍스트 파일

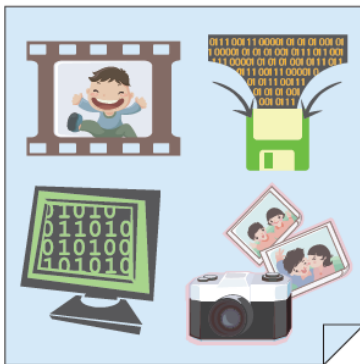
- 예를 들어서 윈도우즈에서는 텍스트 파일이 그림 15-7과 같이 저장된다.



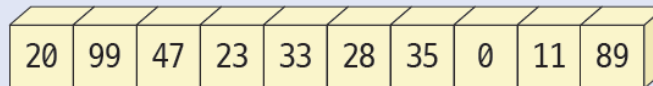
```
Listner - [c:\Users\chun\Documents\Visual Studio 2008\Projects\kkk\kkk\kkk.cpp]
파일(F) 편집(E) 옵션(O) 인코딩(C) 도움말(H) 100 %
00000000: 48 45 4C 4C 4F 0D 0A 57 | 4F 52 4C 44 21 0D 0A 54 | HELLO■■■WORLD!■■■T
00000010: 68 69 73 20 69 73 20 61 | 20 | his is a
줄의 끝을 나타낸다.
```

# 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



이진 파일: 데이터로 구성된 파일



이진 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

파일 열기

예

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

파일 이름

파일 모드

# FILE 구조체

- `fopen( )`은 주어진 파일 이름을 가지고 파일을 생성하여 FILE 포인터를 반환한다.

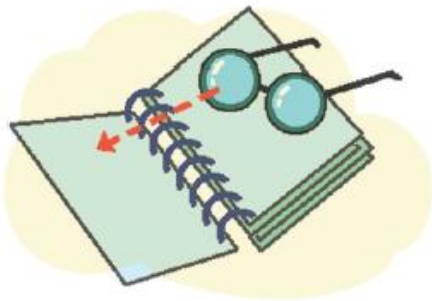
```
struct _iobuf {  
    char *_ptr;  
    int  _cnt;  
    char *_base;  
    int  _flag;  
    int  _file;  
    int  _charbuf;  
    int  _bufsiz;  
    char *_tmpfname;  
};  
typedef struct _iobuf FILE;
```



# 파일 모드

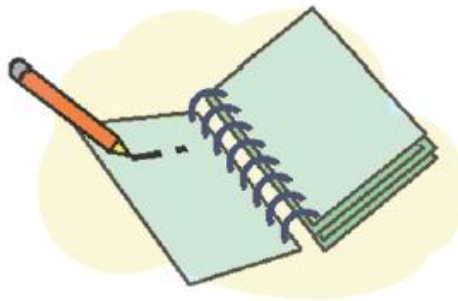
모드	설명
"r"	읽기 모드로 파일을 연다. 만약 파일이 존재하지 않으면 오류가 발생한다.
"w"	쓰기 모드로 새로운 파일을 생성한다. 파일이 이미 존재하면 기존의 내용이 지워진다.
"a"	추가 모드로 파일을 연다. 만약 기존의 파일이 있으면 데이터가 파일의 끝에 추가된다. 파일이 없으면 새로운 파일을 만든다.
"r+"	읽기 모드로 파일을 연다. 쓰기 모드로 전환할 수 있다. 파일이 반드시 존재하여야 한다.
"w+"	쓰기 모드로 새로운 파일을 생성한다. 읽기 모드로 전환할 수 있다. 파일이 이미 존재하면 기존의 내용이 지워진다.
"a+"	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.

# 파일 닫기

Syntax

파일 닫기

예

`fclose(fp):`

FILE 포인터

# 예제

```
#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 삭제 예제

```
#include <stdio.h>

int main(void)
{
    if (remove("sample.txt") == -1)
        printf("sample.txt를 삭제 할 수 없습니다.\n");
    else
        printf("sample.txt를 삭제 하였습니다.\n");
    return 0;
}
```

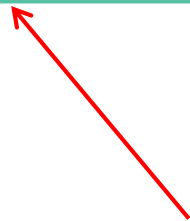
sample.txt를 삭제 하였습니다.

# 기타 유용한 함수들

함수	설명
<code>int foef(FILE *stream)</code>	파일의 끝이 도달되면 true를 반환한다.
<code>int rename(const char *oldname, const char *newname)</code>	파일의 이름을 변경한다.
<code>FILE *tmpfile()</code>	임시 파일을 생성하여 반환한다.
<code>int ferror(FILE *stream)</code>	스트림의 오류 상태를 반환한다. 오류가 발생하면 true가 반환된다.

# 파일 입출력 함수

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



크게 나누면  
텍스트 입출력  
함수와 이진  
데이터  
입출력으로 나눌  
수 있습니다.



# 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;
}
```

파일 열기 성공



# 문자 단위 입출력

```
#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;
}
```

정수형 변수로 선언하여야 한다. 그 이유는 다음 슬라이드에서 설명한다.

파일 열기 성공  
abc

sample - 메모장

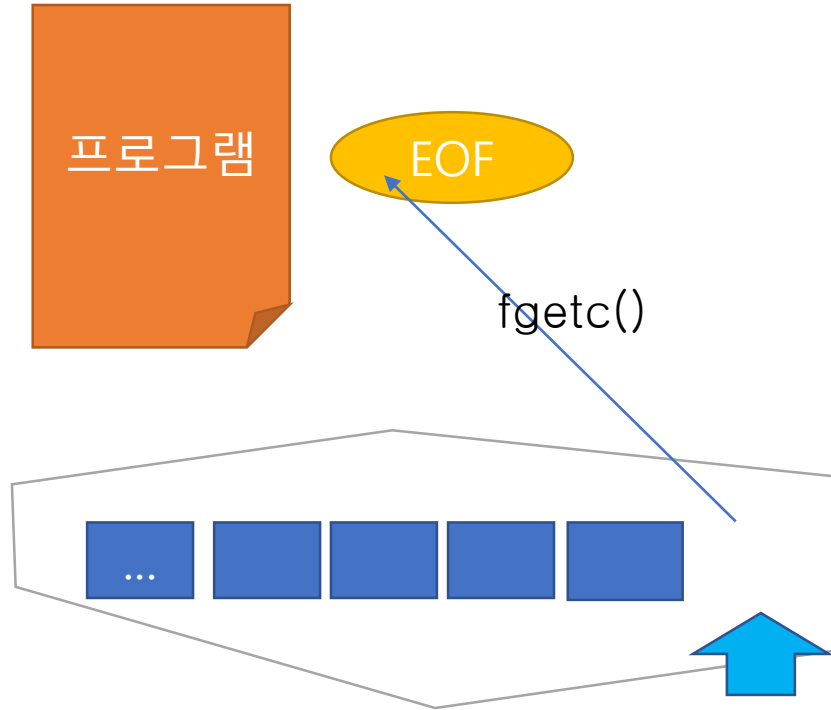
파일(F) 편집(E) 서식(O) 보기(V) 도움말(H)

abc

# EOF

- EOF(End Of File): 파일의 끝을 나타내는 특수한 기호

#define EOF (-1)



# 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

`fgets()` 함수는 최대 `n-1`개의 문자를 `FILE*` 스트림에서 읽되,  
다음 중 하나라도 먼저 발생하면 읽기를 멈춤

1. 줄바꿈 문자 `\n` 을 만났을 때
2. 파일 끝(**EOF**) 에 도달했을 때
3. 버퍼가 가득 찼을 때 (`size - 1`, 마지막은 null character)

# 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: 파일에서 특정 문자열 탐색

- 텍스트 파일에서 특정 문자열을 탐색하는 프로그램을 작성하여 보자. 사용자로부터 입력 텍스트 파일 이름과 탐색할 문자열을 받는다

입력 파일 *proverbs.txt*

```
Absence makes the heart grow fonder.  
Actions speak louder than words.  
All for one and one for all.  
All's fair in love and war.  
...
```

입력 파일 이름을 입력하시오: *proverbs.txt*

탐색할 단어를 입력하시오: *man*

*proverbs.txt: 16 Behind every good man is a good woman.*

*proverbs.txt: 41 A dog is a man's best friend.*

*proverbs.txt: 57 Early to bed and early to rise makes a man healthy, wealthy, and wise.*

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

int main(void)
{
    FILE* fp;
    char fname[128], buffer[256], 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 입력하시요: (음수이면 종료)1 KIM 10.0  
학번, 이름, 성적을 입력하시요: (음수이면 종료)2 PARK 20.0  
학번, 이름, 성적을 입력하시요: (음수이면 종료)3 LEE 30.0  
학번, 이름, 성적을 입력하시요: (음수이면 종료)-1  
평균 = 20.000000



• Text 실습 vsc 내용

[https://github.com/prof-kweon/C-Language-Course/blob/main/3.Practice/file\\_text.c](https://github.com/prof-kweon/C-Language-Course/blob/main/3.Practice/file_text.c)

```

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

    fp = fopen ( "sample.txt" , "w" );
    if ( fp == NULL ) {
        printf ("file opening Failed \n" );
    }
    else {
        printf ("file opening Success \n" );
    }

    // 1. save character
    fputc ('a', fp );
    fputc ('b', fp );
    fputc ('c', fp );
    fputc ('\n', fp );

    // 2. save string
    fputs("hello", fp);
    fputs(" world\n", fp);

    // 3. save number
    fprintf(fp, "%d %d %d %.2f", 1, 2, 3, 3.14);

    fclose ( fp );
    return 0;
}

```

```

int readFile() {
    FILE *fp = fopen("sample.txt", "r"); // read mode

    if (fp == NULL) {
        printf("Fail to open\n");
        return 1;
    }

    // 1. read 4 characters
    char ch1 = fgetc(fp);
    char ch2 = fgetc(fp);
    char ch3 = fgetc(fp);
    char ch4 = fgetc(fp);

    // 2. read string (12 character "hello world\n"
    // including white space)
    char str1[10];
    char str2[10];
    fscanf(fp, "%s %s", str1, str2);

    // 3. read 3 integers & 1 real number
    int n1, n2, n3;
    float f;
    fscanf(fp, "%d %d %d %f", &n1, &n2, &n3, &f);

    // print
    printf("\n== readFile ==\n");
    printf("Chars: %c %c %c %c", ch1, ch2, ch3, ch4);
    printf("String: %s %s\n", str1, str2);
    printf("Integers: %d %d %d\n", n1, n2, n3);
    printf("Float: %.2f\n", f);

    fclose(fp);
    return 0;
}

```

```

int readFileByOne( void )
{
    FILE * fp = NULL;
    int c;
    fp = fopen ( "sample.txt" , "r" );
    if ( fp == NULL )
        printf ( "file opening Failed \n");
    else
        printf ( "file opening Success \n");

    printf("\n== readFileByOne ==\n");
    while ((c = fgetc ( fp )) != EOF ) {
        putchar (c);
    }
    fclose ( fp );
    return 0;
}

```

```

int readFileByLine( void )
{
    FILE * fp = NULL;
    int SIZE = 100;
    char line[SIZE];

    fp = fopen ( "sample.txt" , "r" );
    if ( fp == NULL )
        printf ( "file opening Failed \n");
    else
        printf ( "file opening Success \n");

    printf("\n== readFileByLine ==\n");
    while (fgets (line, SIZE, fp )) {
        printf ("%s", line);
    }
    fclose ( fp );
    return 0;
}

```

```

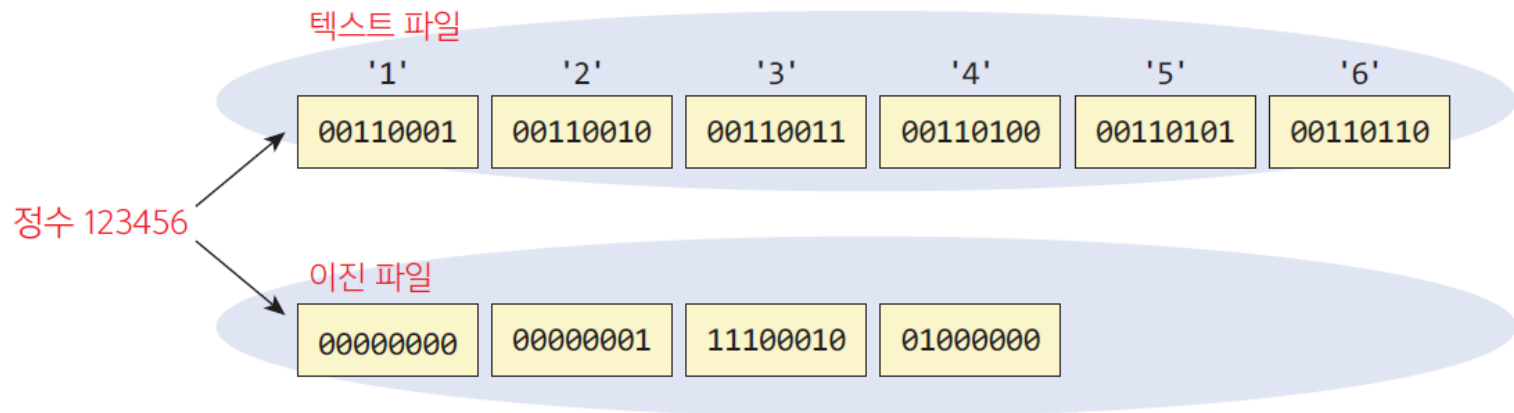
void main()
{
    writeFile();
    readFile();
    readFileByOne();
    readFileByLine();
}

```



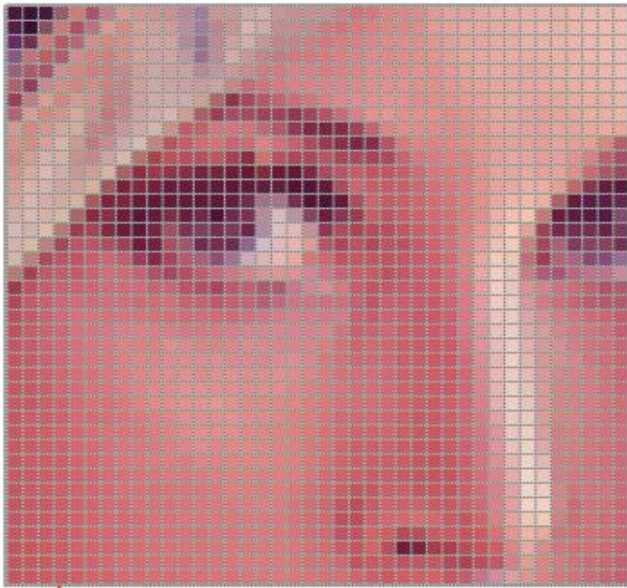
# 이진 파일 쓰기과 읽기

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



# 이진 파일의 예

- 이미지 파일이나 사운드 파일



각 픽셀의 밝기를 이진수로  
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 파일을 열 수 없습니다.");
        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);
```

메모리 블록의 주소

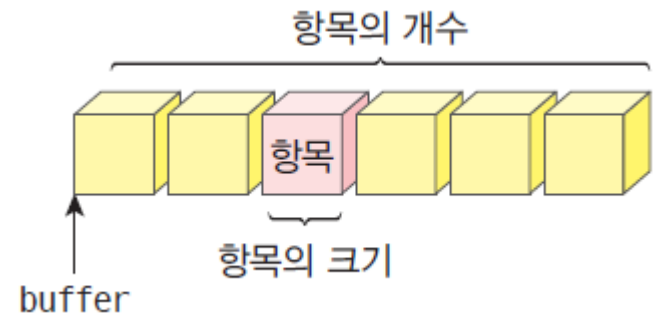
항목의 개수

항목의 크기

FILE 포인터

## `fwrite(buffer, size, count, fp)`

- `buffer`는 파일에 기록할 데이터를 가지고 있는 메모리 블록의 시작 주소이다.
- `size`는 저장되는 항목의 크기로서 단위는 바이트이다.
- `count`는 저장하려고 하는 항목의 개수이다. 만약 `int`형의 데이터 10개를 쓰려고 하면 항목의 크기는 4가 되고 항목의 개수는 10이 될 것이다.
- `fp`는 FILE 포인터이다.



# 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;
}
```



10 20 30 40 50

# 이진 파일 읽기

## Syntax

fread()

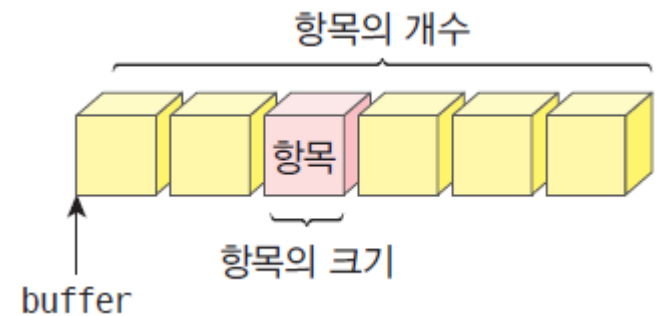
예

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

메모리 블록의 주소      항목의 개수  
항목의 크기      FILE 포인터

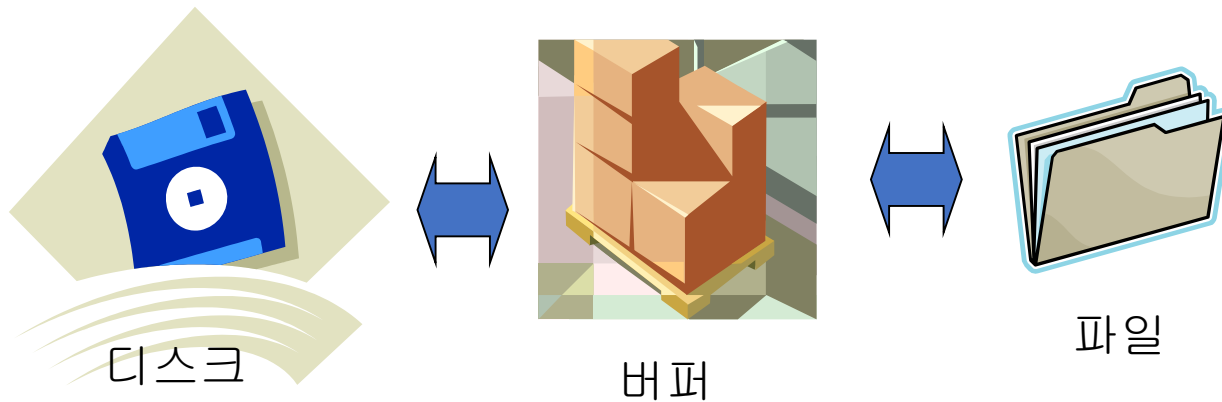
## fread(buffer, size, count, fp)

- **buffer**는 파일에 기록할 데이터를 가지고 있는 메모리 블록의 시작 주소이다.
- **size**는 저장되는 항목의 크기로서 단위는 바이트이다.
- **count**는 저장하려고 하는 항목의 개수이다. 만약 **int**형의 데이터 10개를 쓰려고 하면 항목의 크기는 4가 되고 항목의 개수는 10이 될 것이다.
- **fp**는 FILE 포인터이다.



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



• Binary 실습 vsc 내용

[https://github.com/prof-kweon/C-Language-Course/blob/main/3.Practice/file\\_binary.c](https://github.com/prof-kweon/C-Language-Course/blob/main/3.Practice/file_binary.c)

```

#include <stdio.h>

struct Data {
    char c;
    int i;
    float f;
};

int writeFile() {
    // "wb" = write binary
    FILE *fp = fopen("data.bin", "wb");
    if (fp == NULL) {
        printf("Fail to open\n");
        return 1;
    }

    struct Data d = { 'A', 100, 3.14f };

    // save binary of structure
    fwrite(&d, sizeof(struct Data), 1, fp);

    fclose(fp);
    printf("Complete writing.\n");
    return 0;
}

```

```

int readFile() {
    // "rb" = read binary
    FILE *fp = fopen("data.bin", "rb");

    if (fp == NULL) {
        printf("Fail to open\n");
        return 1;
    }

    struct Data d;

    // read binary of structure
    fread(&d, sizeof(struct Data), 1, fp);

    printf("[read data]\n");
    printf("\tchar: %c\n", d.c);
    printf("\tint: %d\n", d.i);
    printf("\tfloat: %.2f\n", d.f);

    fclose(fp);
    return 0;
}

void main()
{
    int ret = writeFile();
    if(ret == 0){
        ret = readFile();

        if(ret == 0){
            printf("SUCCESS!!");
        } else {
            printf("FAIL!!");
        }
    }
}

```

# Lab: 이미지 파일 복사하기

- 여기서는 이진 파일을 복사하는 프로그램을 작성하여 보자.

이미지 파일 이름: dog.jpg  
copy.jpg로 이미지 파일이 복사됨



# 힌트

- 이진 파일을 읽거나 쓰려면 `fopen()`을 호출할 때 파일 모드에 "b"를 붙이면 된다. 쓰기 전용 파일을 열려면 "wb"로, 읽기 전용 파일을 열려면 "rb"로 하여야 한다.
  - `src_file = fopen("pome.jpg", "rb");`
  - `dst_file = fopen("copy.jpg", "wb");`
- 이진 파일에서 데이터를 읽으려면 `fread()`를 사용한다.
  - `fread(buffer, 1, sizeof(buffer), src_file);`
- 이진 파일에 데이터를 쓰려면 `fwrite()`를 사용한다.
  - `fwrite(buffer, 1, sizeof(buffer), dst_file);`
- `fread()`는 성공적으로 읽은 항목의 개수를 반환한다. 따라서 0이 반환되면 파일의 끝에 도달한 것으로 볼 수 있다.

# 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);

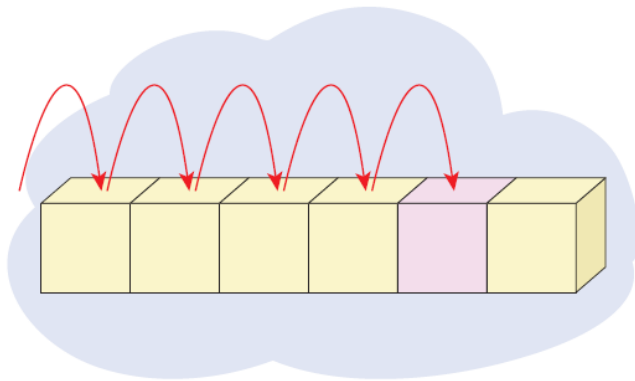
    src_file = fopen(filename, "rb");
    dst_file = fopen("copy.jpg", "wb");
    if (src_file == NULL || dst_file == NULL) {
        fprintf(stderr, "파일 열기 오류 \n");
        return 1;
    }
}
```

# 예제

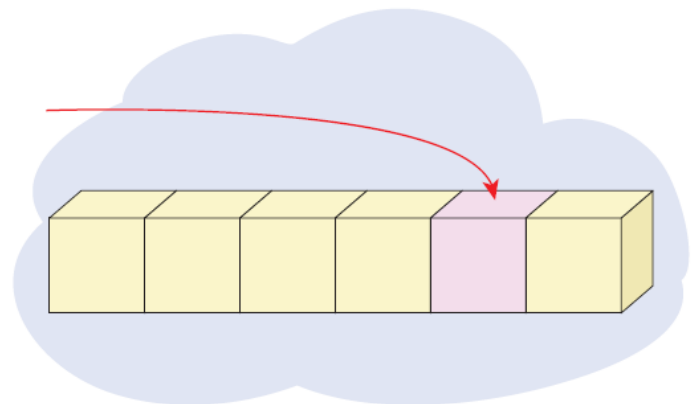
```
while ((r_count = fread(buffer, 1, sizeof(buffer), src_file)) > 0) {  
    int w_count = fwrite(buffer, 1, r_count, dst_file);  
    if (w_count < 0) {  
        fprintf(stderr, "파일 쓰기 오류 \n");  
        return 1;  
    }  
    if (w_count < r_count) {  
        fprintf(stderr, "미디어 쓰기 오류 \n"); 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

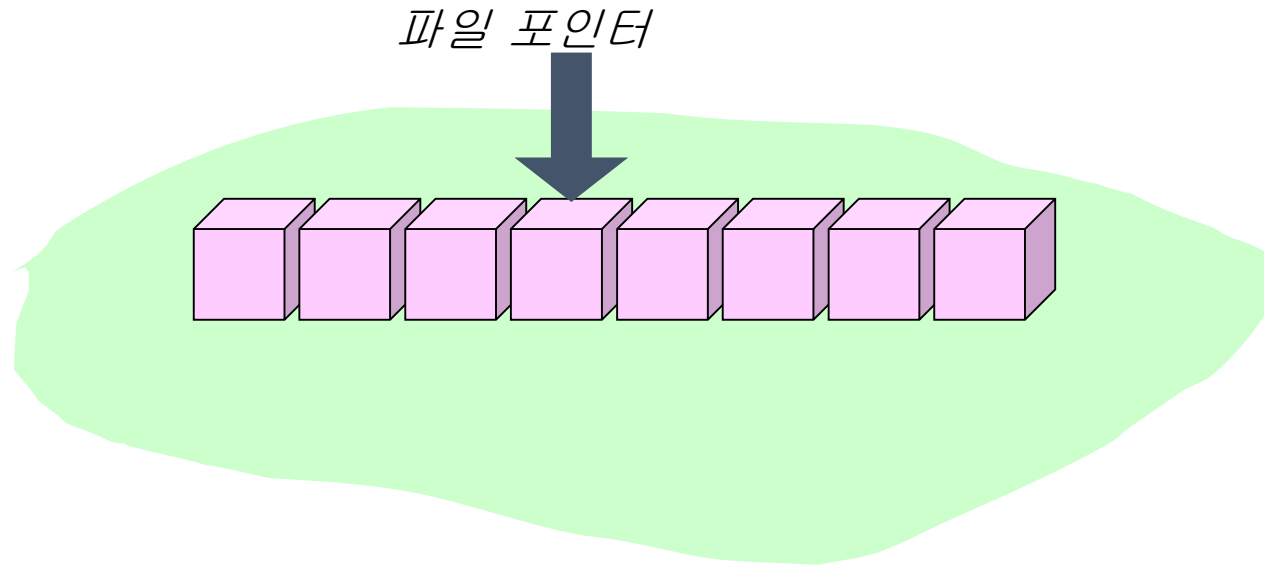


임의 접근파일



# 임의 접근 파일의 원리

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



- 강제적으로 파일 포인터를 이동시키면 임의 접근이 가능

# fseek()

## Syntax

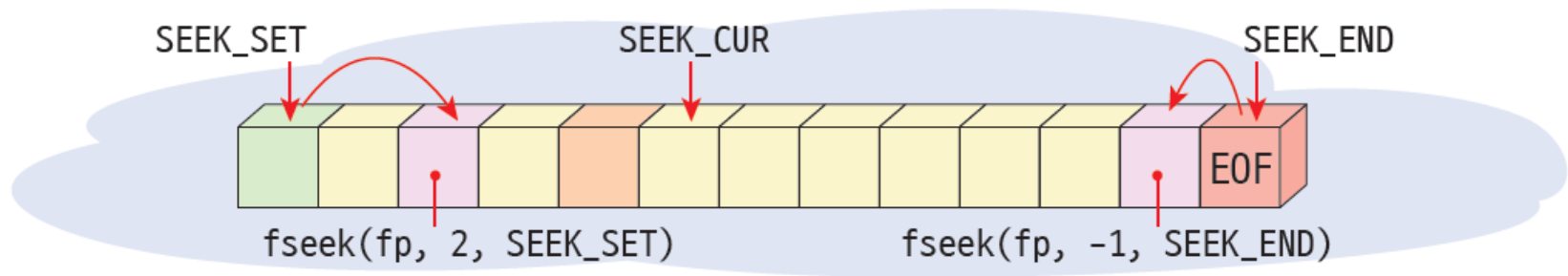
fseek()

예

`int` fseek(FILE 포인터 `FILE *fp`, 거리 `long offset`, 기준 위치 `int origin`);

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

# fseek()



- `rewind(fp)`: 파일 포인터를 첫 부분으로 초기화한다.

# ftell(), feof()

Syntax: ftell()

파일 포인터의 현재의 위치를 반환한다.

예

`long ftell(FILE *fp);`

Syntax: feof()

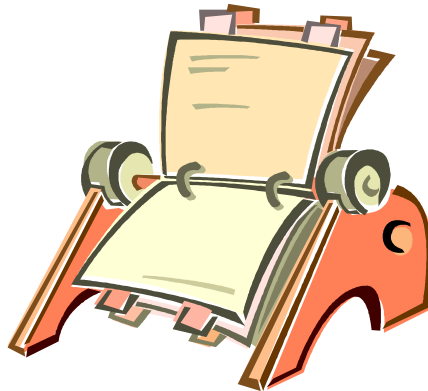
파일의 끝에 도달하였는지 여부를 반환한다.

예

`int feof(FILE *fp);`

# 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

=====

정수값을 입력하시오: 1  
이름: 홍길동  
주소: 서울시 종로구 1번지  
휴대폰: 010-1234-5678  
특징: 초능력 슈퍼 히어로

# 힌트

1. 적합한 것은 “**a+**”모드이다. 주로 추가, 탐색 할 예정
2. 파일에서 읽기 전에 무조건 **fseek()**를 해주어야 한다.
3. 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);
```



```

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("휴대 폰");

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("휴대 폰\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)
{
    //...
}
```

# Q & A

