

프로그래밍연습 Lab 12

파일 입출력

[TA] 강성민, 김기현, 최석원, 최지은, 표지원

Department of Computer Science and Engineering

Seoul National University, Korea

2022/11/23

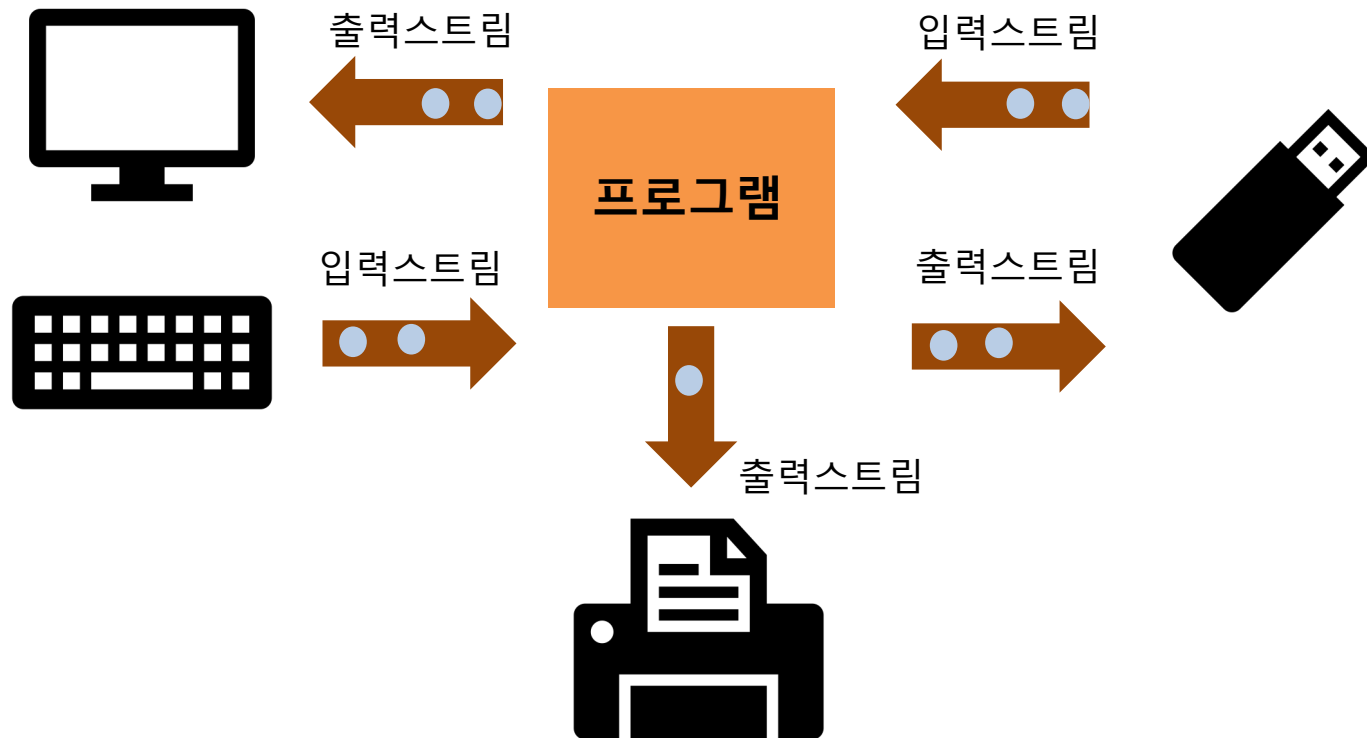
이번 장에서 학습할 내용

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

1. 스트림의 개념

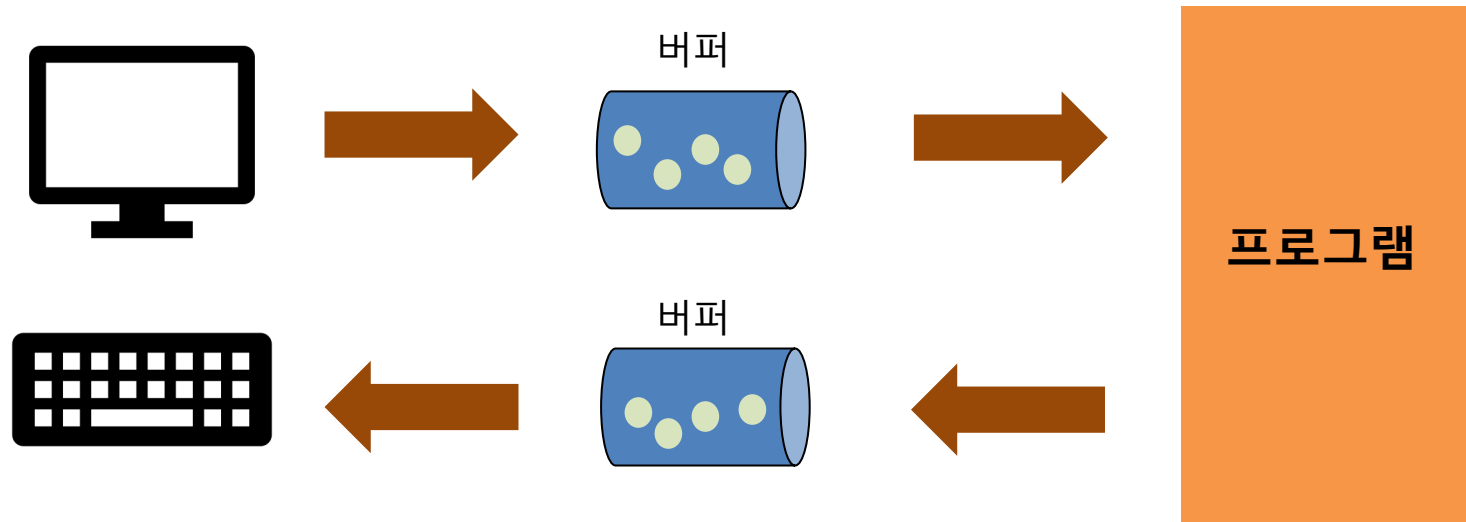
- 스트림(stream)

- 입력과 출력을 바이트(byte)들의 흐름으로 생각하는 것



1. 스트림의 개념

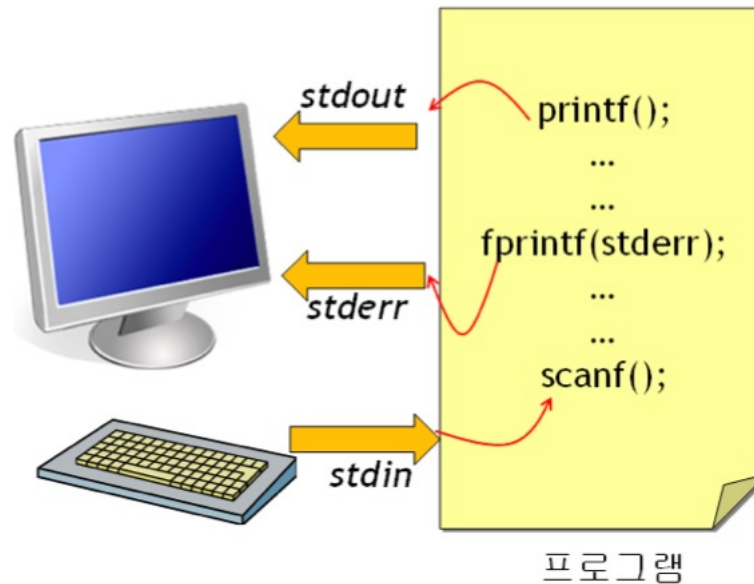
- 스트림(stream)과 버퍼
 - 스트림에는 기본적으로 버퍼가 포함되어 있다.



2. 표준 입출력 스트림

- 기본적인 스트림들은 프로그래머가 생성하지 않아도 자동 생성된다.

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



2. 표준 입출력 스트림

■ 입출력 함수의 분류

- 사용하는 스트림에 따른 분류
 - 표준 입출력 스트림을 사용하여 입출력을 하는 함수
 - 스트림을 구체적으로 명시해주어야 하는 입출력 함수

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

2. 표준 입출력 스트림

■ 입출력 함수의 분류

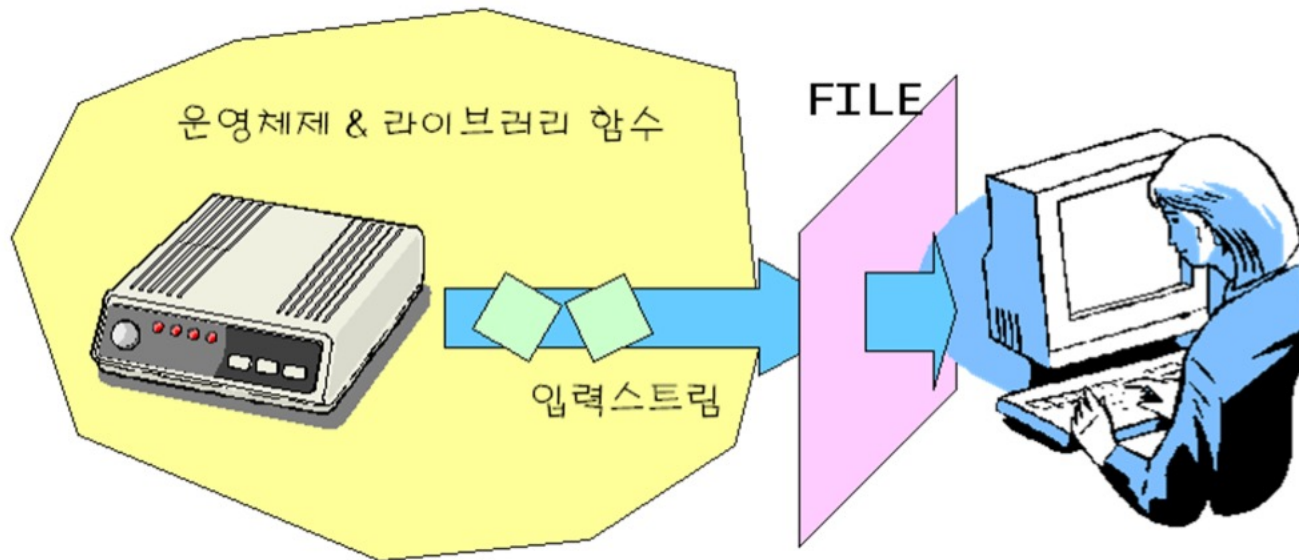
■ 데이터의 형식에 따른 분류

- getchar()나 putchar()처럼 문자 형태의 데이터를 받아들이는 입출력
- printf()나 scanf()처럼 구체적인 형식을 지정할 수 있는 입출력

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

2. 스트림과 파일

- 스트림은 구체적인 FILE 구조체를 통해 구현
- FILE은 stdio.h에 정의되어 있다.



3. 파일 입출력

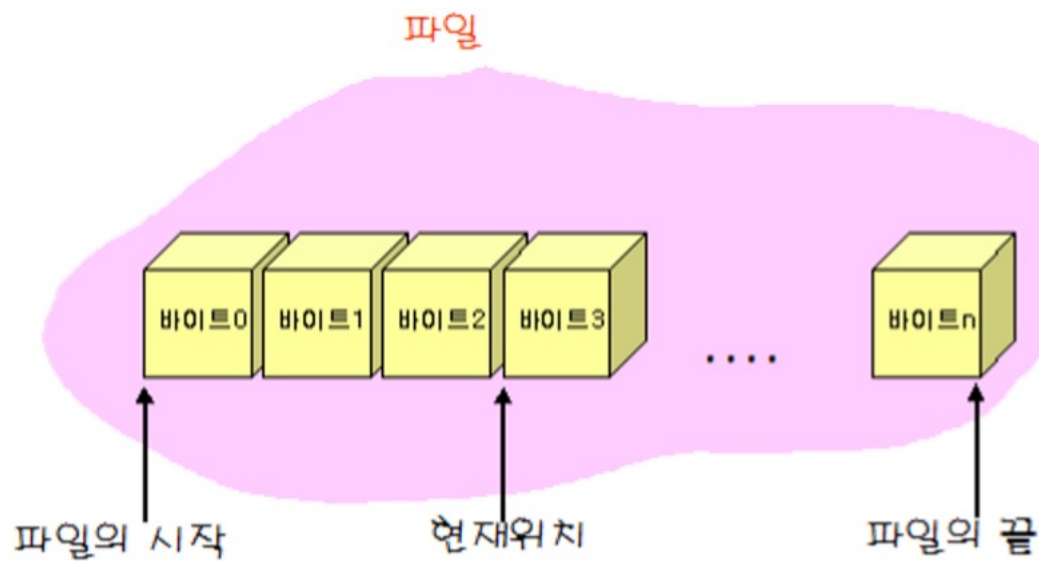
- 파일이 필요한 이유

- 변수, 배열, 구조체 등은 모두 메모리에 만들어지고, 이것들은 모두 전원이 꺼지면 사라진다.
- 하드 디스크에 파일 형태로 저장하면 전원이 꺼져도 데이터가 보존됨.

- 파일

- C에서 파일은 일련의 연속된 바이트
- 모든 파일 데이터들은 결국 바이트로 바뀌어서 파일에 저장
- 이 바이트들을 어떻게 해석하는지는 프로그래머의 책임

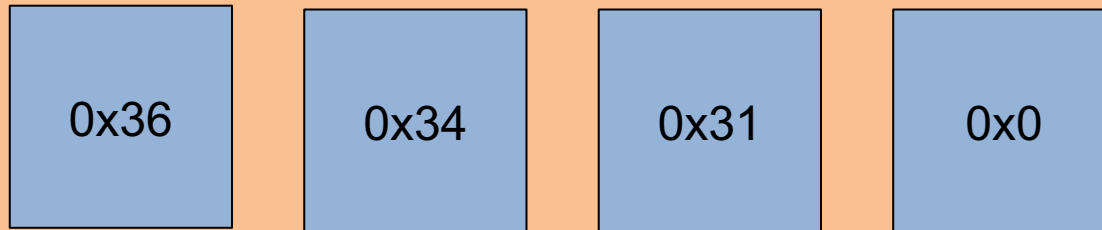
3. 파일 입출력



3. 파일 입출력

- 파일

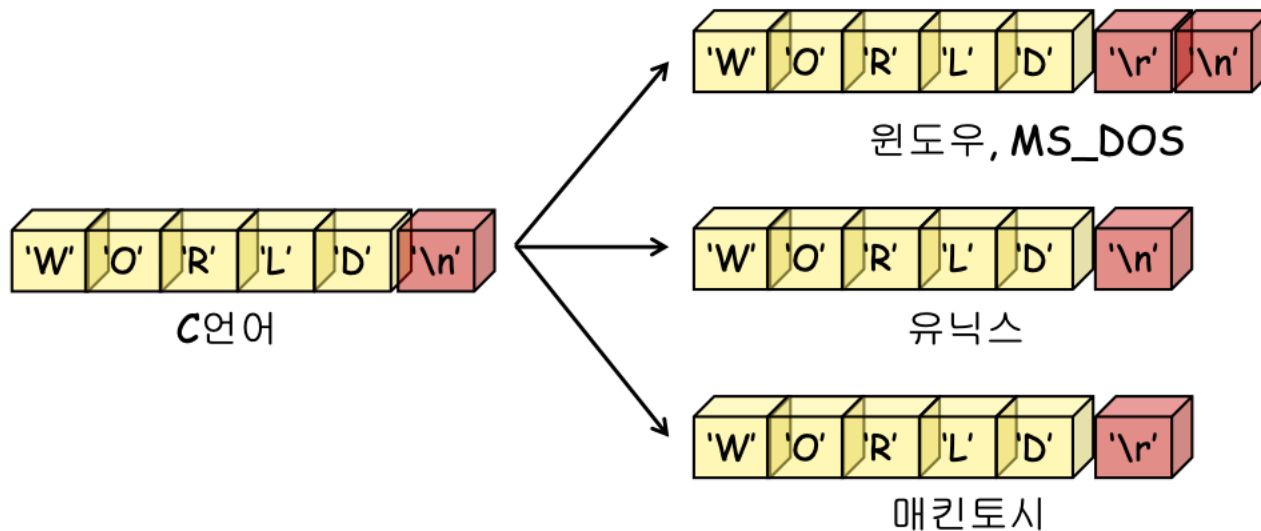
- 파일에 4개의 바이트가 들어 있을 때 이것을 Int형 정수 데이터로도 해석할 수 있고, float형 실수 데이터, 혹은 4개의 문자로도 해석 가능하다.



3. 파일 입출력

■ 텍스트 파일

- 텍스트 파일은 사람이 읽을 수 있는 텍스트가 들어있는 파일.
 - ex) c프로그램 소스파일이나 메모장 파일
- 텍스트 파일은 아스키 코드를 이용하여 저장
- 텍스트 파일은 연속적인 라인들로 구성



3. 파일 입출력

■ 파일 처리의 개요

- 디스크 파일은 FILE 구조체를 이용해 접근
- **파일 포인터**: FILE 구조체를 가리키는 포인터
- 파일을 다룰 때에는 반드시 다음과 같은 순서를 지켜야 한다.
- **파일 열기 -> 파일 읽기와 쓰기 -> 파일 닫기**

■ 파일 열기

- 파일에서 데이터를 읽거나 쓸 수 있도록 모든 준비를 마치는 것

```
FILE *fopen(const char *name, const char *mode)
```

- 첫 번째 매개 변수인 name은 파일 이름
- 두 번째 매개 변수인 mode는 파일을 여는 모드를 의미.

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

3. 파일 입출력

■ 파일 모드

모드	설명
"r"	읽기 모드로 파일을 연다.
"w"	쓰기 모드로 파일을 생성한다. 만약 파일이 존재하지 않으면 파일이 생성된다. 파일이 이미 존재하면 기존의 내용이 지워진다.
"a"	추가 모드로 파일을 연다. 만약 똑같은 이름의 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 파일을 생성한다. 읽기 모드로 전환할 수 있다. 파일이 존재하면 기존의 데이터가 지워진다.
"a+"	추가 모드로 파일을 연다. 읽기 모드로 전환할 수 있다. 데이터를 추가하면 EOF 마커를 추가된 데이터의 뒤로 이동한다. 파일이 없으면 새로운 파일을 만든다.
"b"	바이너리 파일 모드로 파일을 연다.

3. 파일 입출력

■ 파일 모드

- 기본적인 파일 모드에 “+”나 “b”를 붙일 수 있음 (wb, r+)
- “a”나 “a+”모드는 추가(append)모드.
- 추가모드로 파일이 열리면, 모든 쓰기 동작은 파일의 끝에서 일어나며 기존 데이터는 절대 지워지지 않음
- “r+”, “w+”, “a+” 파일 모드가 지정되면 읽고 쓰기가 모두 가능
- 수정 (update) 모드. 읽기 모드에서 쓰기모드로, 또는 쓰기 모드에서 읽기 모드로 전환하려면 반드시 fflush(), fsetpos(), fseek(), rewind()중 하나를 호출해야 한다.

예제 1. 파일 열기 (file_open.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

3. 파일 입출력

- 파일을 닫는 함수

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

- 파일을 삭제하는 함수 (예제: file_close.c)

```
int remove(const char *path)
```

```
#include <stdio.h>

int main( void )
{
    if( remove( "sample.txt" ) == -1 )
        printf( "sample.txt를 삭제할 수 없습니다.\n" );
    else
        printf( "sample.txt를 삭제하였습니다.\n" );

    return 0;
}
```

2012 생능출판사 All rights reserved

3. 파일 입출력

■ 파일 입출력 라이브러리 함수

종류	설명	입력 함수	출력 함수
문자 단위	문자 단위로 입출력	int fgetc(FILE *fp)	int fputc(int c, FILE *fp)
문자열 단위	문자열 단위로 입출력	char *fgets(FILE *fp)	int fputs(const char *s, FILE *fp)
서식화된 입출력	형식 지정 입출력	int fscanf(FILE *fp, ...)	int fprintf(FILE *fp,...)
이진 데이터	이진 데이터 입출력	fread()	fwrite()

• 문자 입출력 함수

```
int fgetc( FILE *fp );
```

```
int fputc( int c, FILE *fp );
```

FILE

파일 포인터

• 문자열 입출력 함수

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

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

문자열의 크기


FILE INPUT

3. 파일 입출력

- 한 글자씩 쓰기 (fputc.c: file_open.c 내용 복사하여 수정)
- 1) 문자 단위 입력(파일 안에 그 결과를 출력)

```
#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;
}
```

sample.txt


abc

파일 열기 성공

3. 파일 입출력

- 한 글자씩 읽기 (fgetc.c: file_open.c 내용 복사하여 수정)
- 2)문자 단위 출력(파일내용을 입력받는다==가져온다)

```
#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");

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

sample.txt

abc

파일 열기 성공
abc

실습 1. 텍스트 파일 복사하기(copytxt.c)

- 문자열 입출력 함수(fgetc(), fputc())를 이용해 텍스트 파일 복사
- 원본 파일과 복사 파일의 이름을 입력 받은 후 원본 파일에서 한 문자씩 읽어서 문자형 변수에 저장
- 저장된 문자는 다시 복사 파일로 출력

Algorithm

- 원본 파일을 읽기 모드로 연다. 반환값이 NULL이면 오류 메시지 출력, exit(1) 호출하여 프로그램 종료
- 같은 방법으로 복사 파일을 쓰기 모드로 연다. 동일한 이름이 디스크에 존재하면 기존내용은 삭제된다.
- fgetc()를 호출하여 원본 파일에서 문자를 읽어 변수 c로 복사.
- 읽은 문자를 fputc()로 복사 파일에 기록한다.

원본 파일 이름: a.txt
복사 파일 이름: b.txt

실습 1. 텍스트 파일 복사하기

```
#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);
    }
```

실습 1. 텍스트 파일 복사하기

```
// 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

실습 2. 파일 내용에서 단어 검색

■ 문자열 입력 함수 fgets() 이용

- 문자열 입출력 함수

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

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

문자열의 크기

FILE INPUT

■ strstr() 이용

- 문자열 비교 함수로, str1과 str2에 같은 내용이 있는지 확인함
- str1에 str2의 문자열과 일치하는 문자열이 있으면 해당 위치의 포인터(char* 타입)를 반환
- 일치하는 문자열을 찾지 못하면 null pointer반환
(따라서 NULL필수)

```
char * strstr(char* str1, const char* str2);
```

실습 2. 파일 내용에서 단어 검색

- proverb.txt 내용

A chain is only as strong as its weakest link
A change is as good as a rest
A fool and his money are soon parted
A friend in need is a friend indeed
A good beginning makes a good ending
A little knowledge is a dangerous thing

```
ord && "/Users/macpro/Downloads/C/"findword
입력 파일 이름 : proverb.txt
탐색할 단어 입력 : in
proverb.txt: line 1에서 in 발견 .
proverb.txt: line 4에서 in 발견 .
proverb.txt: line 5에서 in 발견 .
proverb.txt: line 6에서 in 발견 .
```

실습 2. 파일 내용에서 단어 검색

- 코드

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

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

    printf("입력 파일 이름: ");
    scanf("%s", fname);
    printf("탐색할 단어 입력: ");
    scanf("%s", word);
```

실습 2.파일 내용에서 단어 검색

■ 코드

```
if( (fp =fopen(fname, "r"))== NULL) {  
    fprintf(stderr, "파일 %s를 열 수 없음.\n", fname);  
    exit(1);  
}  
while( fgets(buffer, 256, fp)) {  
    line_num++;  
    if(strstr(buffer, word)){  
        printf("%s: line %d에서 %s 발견. \n",fname,line_num,word);  
    }  
}  
fclose(fp);  
return 0;
```

```
}
```

3. 파일 입출력

■ 형식화된 입력

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

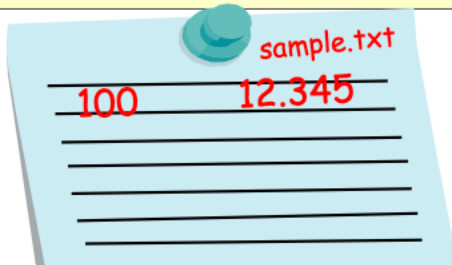
```
int main(void)
{
    int i;
    float f;
    FILE *fp;

    fp = fopen("sample.txt", "r");
    if( fp != NULL )
        fscanf(fp, "%d %f", &i, &f);
    printf("%d %f", i, f);
    fclose(fp);
}
```



100	12.345
-----	--------

%d와 같은
특정한 형식을
지정하여 파일에
입력할 수 있습니다.



예제 2. 파일에 성적 입력, 평균 구하기

```
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 2. 파일에 성적 입력, 평균 구하기

```
// 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\n", 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 2. 파일에 성적 입력, 평균 구하기

```
// 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: **score.txt**

학번, 이름, 성적을 입력하시요: (음수이면 종료) 1 KIM 90.2

학번, 이름, 성적을 입력하시요: (음수이면 종료) 2 PARK 30.5

학번, 이름, 성적을 입력하시요: (음수이면 종료) 3 MIN 56.8

학번, 이름, 성적을 입력하시요: (음수이면 종료)-1

평균 = 58.575001

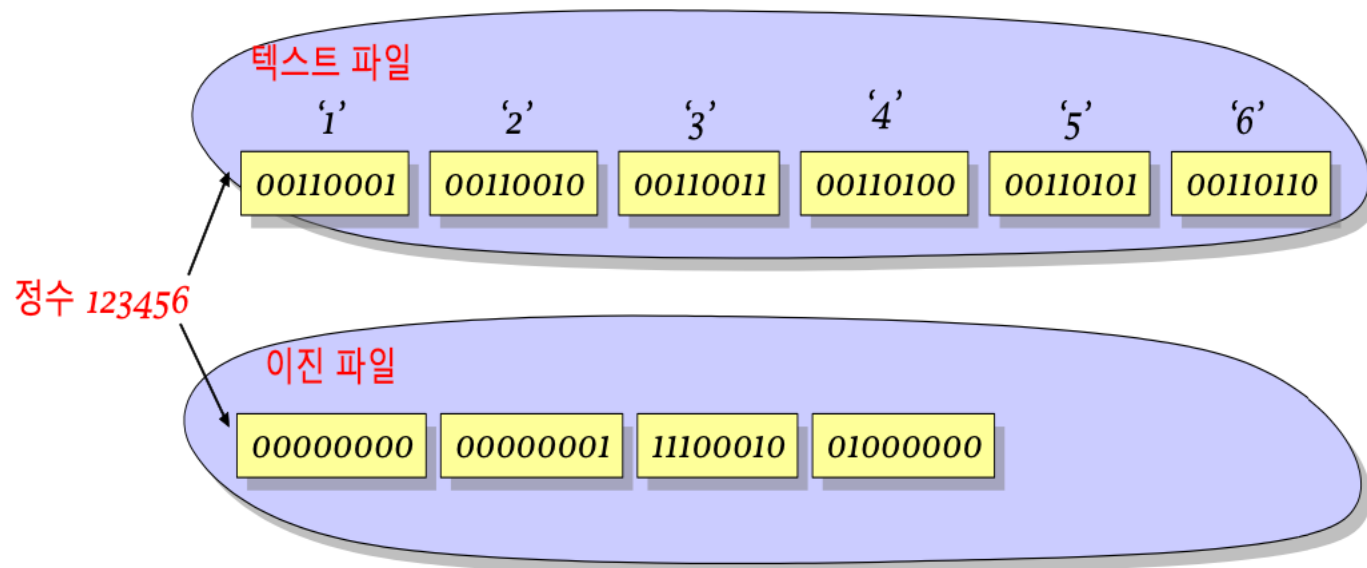
중간 점검

- 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.

바이너리 파일 쓰기과 읽기

- 텍스트 파일과 바이너리 파일의 차이점

- 텍스트 파일: 모든 데이터가 아스키 코드로 변환되어 저장
- 바이너리 파일: 컴퓨터에서 데이터를 표현하는 방식 그대로 저장



바이너리 파일

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

```
int main(void)
{
    FILE *fp = NULL;

    fp = fopen("binary.txt", "rb");

    if( fp == NULL )
        printf("이진 파일 열기에 실패하였습니다.\n");
    else
        printf("이진 파일 열기에 성공하였습니다.\n");

    if( fp != NULL ) fclose(fp);
}
```


바이너리 파일

■ 바이너리 파일 쓰기

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

```
int main(void)
```

```
{
```

```
    int buffer[] = { 10, 20, 30, 40, 50 };
```

```
    FILE *fp = NULL;
```

```
    size_t i, size, count;
```

```
    fp = fopen("binary.bin", "wb");
```

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

```
    {
```

```
        fprintf(stderr, "binary.txt 파일을 열 수 없습니다.");
```

```
        exit(1);
```

```
    }
```

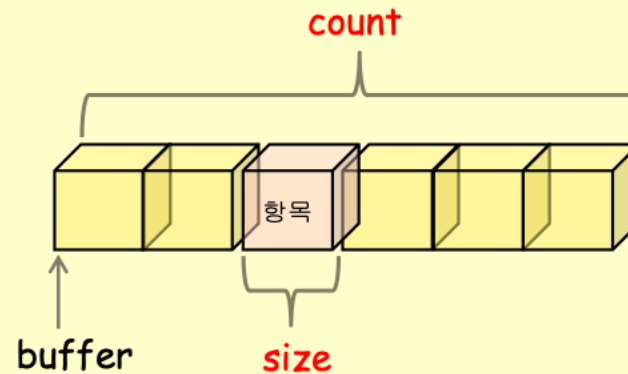
```
    size = sizeof(buffer[0]);
```

```
    count = sizeof(buffer) / sizeof(buffer[0]);
```

```
    i = fwrite(&buffer, size, count, fp);
```

```
    return 0;
```

```
}
```



바이너리 파일

■ 바이너리 파일 읽기

```
#include <stdio.h>
#define SIZE 1000
```

```
int main(void)
```

```
{
```

```
    float buffer[SIZE];
```

```
    FILE *fp = NULL;
```

```
    size_t size;
```

```
    fp = fopen("binary.txt", "rb");
```

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

```
    {
```

```
        fprintf(stderr, "binary.txt 파일을 열 수 없습니다.");
```

```
        exit(1);
```

```
    }
```

```
    size = fread( &buffer, sizeof(float), SIZE, fp);
```

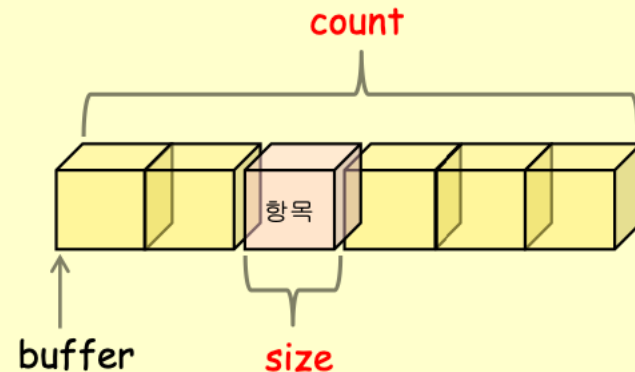
```
    if( size != SIZE )
```

```
        fprintf(stderr, "읽기 동작 중 오류가 발생했습니다.\n");
```

```
    fclose(fp);
```

```
    return 0;
```

```
}
```



버퍼링

- `fopen()`으로 파일을 열면, 버퍼가 자동으로 생성됨.
- 버퍼는 파일로부터 읽고 쓰는 데이터의 **임시 저장소**로 이용되는 메모리의 블록임 -> `fflush(fp)`명령어로 버퍼의 내용을 디스크로 내보냄
- 디스크 드라이브는 블록 단위 장치이므로, 블록단위 입출력이 가장 효율적. (1024바이트의 블록)
- 파일과 연결된 버퍼는 파일과 물리적인 디스크 사이의 인터페이스로 사용된다.

예제 3. 이진파일 입출력 예제

■ 학번, 이름, 평점 출력

```
#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 3. 이진파일 입출력 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
```

예제 4. 이진파일 입출력 예제

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

int main(void)
{
    FILE *fp1, *fp2;
    char file1[100], file2[100];
    char buffer[1024];
    int count;

    printf("첫번째 파일 이름: ");
    scanf("%s", file1);
    printf("두번째 파일 이름: ");
    scanf("%s", file2);
    // 첫번째 파일을 쓰기 모드로 연다.
    if( (fp1 = fopen(file1, "rb")) == NULL )
    {
        fprintf(stderr, "입력을 위한 파일을 열 수 없습니다.\n");
        exit(1);
    }
```

예제 4. 이진파일 입출력 예제

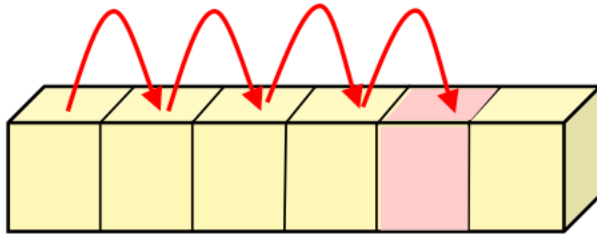
```
// 두번째 파일을 추가 모드로 연다.  
if( (fp2 = fopen(file2, "ab")) == NULL )  
{  
    fprintf(stderr, "추가를 위한 파일을 열 수 없습니다.\n");  
    exit(1);  
}  
// 첫번째 파일을 두번째 파일 끝에 추가한다.  
while((count = fread(buffer, sizeof(char), 1024, fp1)) > 0)  
{  
    fwrite(buffer, sizeof(char), count, fp2);  
}  
fclose(fp1);  
fclose(fp2);  
  
return 0;  
}
```

첫번째 파일 이름: a.dat
두번째 파일 이름: b.dat

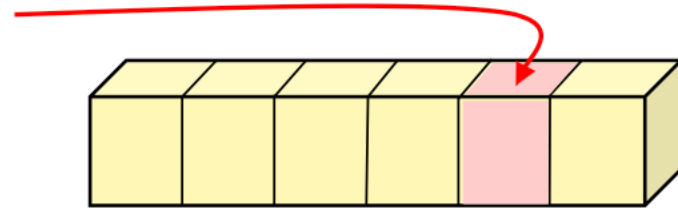
임의 접근

- 순차 접근과 임의 접근

- 순차 접근: 데이터를 파일의 처음부터 순차적으로 읽거나 기록하는것.
- 임의 접근: 파일의 어느 위치에서든지 읽기와 쓰기가 가능



순차접근파일



임의접근파일

임의 접근

■ 임의 접근의 원리

- 파일 포인터: 64비트의 값으로 읽기와 쓰기 동작이 현재 어떤 위치에서 일어나는지 나타냄.
- 새 파일->포인터위치 0.
- 기존파일의 경우 추가모드->파일의 끝.
- 다른 모드인 경우는 파일의 시작부분.
- 파일 읽기/쓰기 수행 시 파일포인터 갱신.
- 강제로 위치 표시자를 조작하면 임의접근 가능
- `fseek()` 함수

임의 접근

- 임의접근 관련 함수
 - fseek()

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

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

```
fseek(fp, 0L, SEEK_SET);           // 파일의 처음으로 이동
fseek(fp, 0L, SEEK_END);           // 파일의 끝으로 이동
fseek(fp, 100L, SEEK_SET);          // 파일의 처음에서 100바이트 이동
fseek(fp, 50L, SEEK_CUR);           // 현재 위치에서 50바이트 이동
fseek(fp, -20L, SEEK_END);          // 파일의 끝에서 20바이트 앞으로 이동
fseek(fp, sizeof(struct element), SEEK_SET); // 구조체만큼 앞으로 이동
```

임의 접근

- 임의접근 관련 함수

- `rewind()`: 파일위치 표시자를 0으로 초기화
- `ftell()`: 파일위치 표시자의 현재위치 반환.

```
void rewind(FILE *fp);
```

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


임의 접근

- 임의 접근이 필요한 경우
 - 음악파일 중간 건너뛰고 뒷부분으로 가고 싶을때.
 - 멀티미디어 파일 크기 너무 커서 메모리에 모두 적재 못 함.
 - -> 파일의 위치 표시자를 이동시키자.

예제 5. 텍스트 파일 중간부분 변경하기

```
C fseek_ex.c > main(void)
1  #include <stdio.h>
2  #include <stdlib.h>
3
4  int main(void){
5      FILE *fp;
6      fp = fopen("data.txt", "w");
7      if (fp == NULL) {
8          printf("data.txt파일을 열 수 없습니다.");
9          exit(1);
10     }
11
12     fputs("This is an house.", fp);
13     fseek(fp, 11, SEEK_SET);
14     fputs("apple", fp);
15     fclose(fp);
16
17     return 0;
18 }
```

≡ data.txt

1 This is an apple.

THANK YOU!