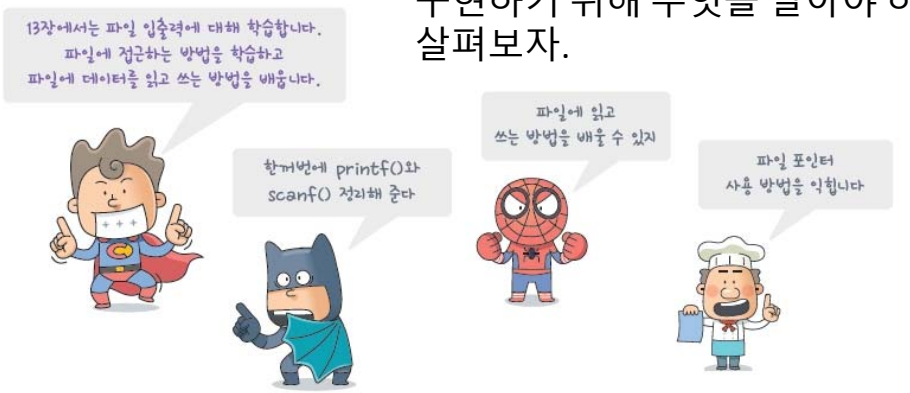
CHAPTER

13 파일 입출력

학습목차

학습목표

- 프로그램이 처리하는 일반적인 데이터는 대부분 파일의 형태로 제공된다. 따라서 파일로 존재하는 데이터를 읽고 쓰는 방법을 모른다면 잘 만든 프로그램도 쓸모가 없다.
- 이제 파일 입출력이 무엇이고 제대로 구현하기 위해 무엇을 알아야 하는지 살펴보자.

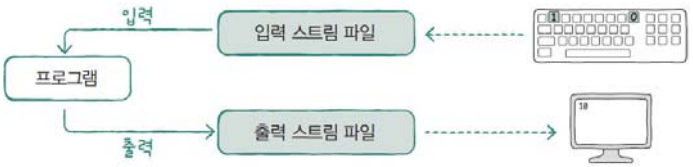


13.1 기본 입출력

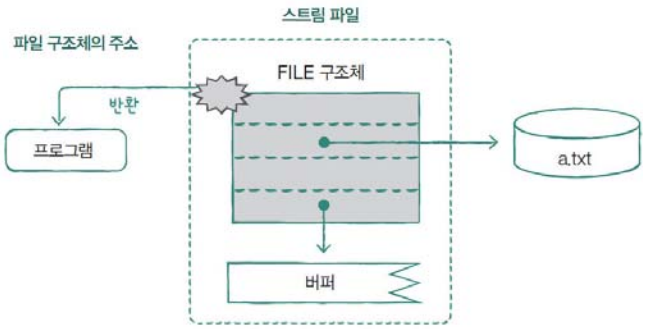
1 입출력 이해하기

- 많은 입출력 장치마다 입력 방식과 출력 방식이 다르면 장치들에게 맞는 입출력 방식을 따로 배워야 하는데 이는 어려움
- 입출력 장치들에게 일관되게 접근 할 수 있는 방법이 있음
- 입출력 장치에 오고가는 데이터는 스트림(stream) 형태
- 컴퓨터 내의 모든 입출력 장치에서 오고가는 데이터는 길게 한 줄로 늘어서 있음
- 입력 장치에서 프로그램으로 들어오는 데이터나 출력 장치로 내보내는 데이터 모두 스트림 형태

- **스트림 파일과 파일 포인터**
  - 파일 입출력은 스트림 파일을 통해서 수행한다.



  - 파일을 개방하면 스트림 파일을 만들고 파일 포인터를 반환한다.



13.1 기본 입출력

2 기본 출력 함수 : printf()

- printf()의 형식과 특성에 대하여 정리

```
printf() 형식 구조      printf(" %(형식) ", var);
```

표 13-1 주요 서식문자

서식문자	자료형	비고
%d	정수(int)	부호가 있는 10진 정수
%f	실수(float)	float와 double을 같이 사용
%c	문자(char)	문자 한 글자(아스키 코드 값)
%s	문자열(char *)	문자 배열 출력(문자열 끝에 NULL이 있어야 함)
%p	포인터(주소)	포인터가 가지고 있는 주소 출력

13.1 기본 입출력

표 13-2 정수관련 서식문자

서식문자	비고	서식문자	비고
%d	부호 있는 10진 정수	%u	unsigend 10진 정수
%i	부호 있는 10진 정수 = %d	%o	8진수 형식으로 출력
%x	16진수 형식으로 출력(소문자)	%X	16진수 형식으로 출력(대문자)
%ld	부호 있는 long	%lu	unsigend long
%lld	부호 있는 long long	%llu	unsigend long long

표 13-3 실수 관련 서식문자

서식문자	의미	비고
%f	소수점으로 출력(소문자)	소수점 이하 6자리까지 출력
%Lf	long double	소수점 이하 6자리까지 출력
%F	소수점으로 출력(대문자)	소수점 이하 6자리까지 출력
%e	지수 표기법으로 출력(소문자)	long double의 경우 %Le
%E	지수 표기법으로 출력(대문자)	long double의 경우 %LE
%g	%f와 %e 중 짧은 것으로 출력(소문자)	
%G	%f와 %e 중 짧은 것으로 출력(대문자)	
%a	16진법으로 표시(소문자)	
%A	16진법으로 표시(대문자)	

13.1 기본 입출력

printf() 서식문자 구조

```
printf(" %-10.3f ", var);
```

(플래그) (출력 길이) (소수점 이하 길이)

표 13-4 플래그

플래그	의미	비고
-	좌측 정렬	기본 좌측 정렬, 글자 폭 지정 우측 정렬
+	숫자 출력 시 +와 - 기호 사용	+ 플래그가 없을 경우 음수(-) 기호만 표시
0	숫자 앞 빈칸을 0으로 채운다	
빈칸	+ 기호 자리에 공백을 넣는다	
#	8진수 앞에 0, 16진수 앞에 0x 붙임	

13.1 기본 입출력

표 13-5 플래그 출력 예

플래그	출력 결과	비고
printf("%d", 123)	1 2 3	기본: 좌측 정렬
printf("%10d", 123)	␣ ␣ ␣ ␣ ␣ ␣ ␣ 1 2 3	길이 지정: 우측 정렬
printf("%010d", 123)	0 0 0 0 0 0 0 1 2 3	숫자 앞 빈칸 0 채움
printf("%-10d", 123)	1 2 3 ␣ ␣ ␣ ␣ ␣ ␣ ␣	- 플래그: 좌측 정렬
printf("%d", -123)	- 1 2 3	기본: 음수는 - 표시
printf("% d", 123)	␣ 1 2 3	빈칸: + 자리 공백으로
printf("%+d", 123)	+ 1 2 3	+ 기호 표시
printf("%#x", 0x12)	0 x 1 2	16진수 앞에 0x 붙임
printf("%10s", "ab")	␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣ a b	길이 지정: 우측 정렬
printf("%-10s", "ab")	a 0 ␣ ␣ ␣ ␣ ␣ ␣ ␣ ␣	- 플래그: 좌측 정렬
printf("%f", 1.23456789)	1 . 2 3 4 5 6 8	소수점 이하 6자리
printf("%.3f", 1.23456789)	1 . 2 3 5	소수점 이하 3자리
printf("%10.3f", 1.23456789)	␣ ␣ ␣ ␣ ␣ 1 . 2 3 5	길이 지정: 우측 정렬
printf("%-10.3f", 1.23456789)	1 . 2 3 5 ␣ ␣ ␣ ␣ ␣	- 플래그: 좌측 정렬

13.1 기본 입출력

[예제 13-1] 여러 형식 지정 출력 코드

```c
#include <stdio.h>

int main() {

   printf("[%d]\n", 123);
   printf("[%10d]\n", 123);
   printf("[%010d]\n", 123);
   printf("[%-10d]\n", 123);
   printf("[%d]\n", -123);
   printf("[% d]\n", 123);
   printf("[%#x]", 0x12);

   printf("[%10s]\n", "ab");
   printf("[%-10s]\n", "ab");

   printf("[%f]\n", 1.23456789);
   printf("[%.3f]\n", 1.23456789);
   printf("[%10.3f]\n", 1.23456789);
   printf("[%-10.3f]\n", 1.23456789);

   return 0;
}
```

실행 화면

```
[123]
[       123]
[0000000123]
[123       ]
[-123]
[ 123]
[0x12]
[        ab]
[ab        ]
[1.234568]
[1.235]
[     1.235]
[1.235     ]
```

3 기본 입력 함수 : scanf()

- scanf()의 형식과 특성에 대하여 정리

scanf() 형식 구조

```
scanf(" %(형식) ", &var);
```

```
01  scanf("%d%d", &num1 &num2);      정상: 공백으로 입력 구분
02  scanf("%d %d\n", &num1 &num2);   비정상: \n을 인식 못함
03  scanf("%c%c", &c1 &c2);          비정상: 공백이 c2에 들어감
```


[예제 13-2] 여러문자 입력 코드

```
01  #include <stdio.h>
02
03  int main() {
04      char c1, c2, c3;
05
06      printf("문자 입력: ");
07      scanf("%c%c%c", &c1, &c2, &c3);
08
09      printf("문자1:[%c]\n", c1);
10      printf("문자2:[%c]\n", c2);
11      printf("문자3:[%c]\n", c3);
12
13      return 0;
14  }
```

실행 화면

```
문자 입력: a b c
문자1:[a]
문자2:[ ]
문자3:[b]
```

13.1 기본 입출력

[예제 13-3] 여러 문자 입력 코드: 문제점 찾아 고치기

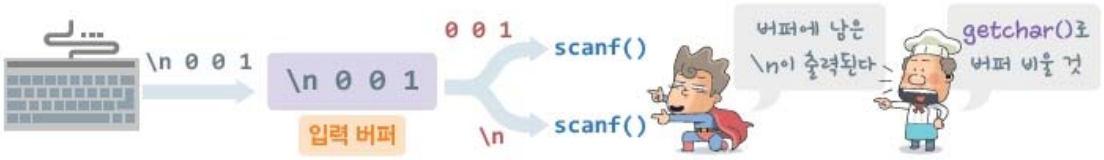
```c
#include <stdio.h>

int main() {
   int num1;
   char c1;

   printf("숫자 입력: ");
   scanf("%d", &num1);
   printf("숫자:[%d]\n", num1);

   printf("문자 입력: ");
   scanf("%c", &c1);
   printf("문자:[%c]\n", c1);

   return 0;
}
```

실행 화면

```
숫자 입력: 100
숫자:[100]
문자 입력: 문자:[
]
```


13.1 기본 입출력

- scanf()와 버퍼 문제



- scanf()를 연달아 사용하는 경우, getchar()를 사용하여 버퍼에 있는 내용을 비워야 한다.

```c
while (getchar() != '\n');
```

버퍼에 남은 것을 비운다

11 %f를 아무것도 지정하지 않는 경우 소수점 몇 번째 자리까지 출력되는가?

① 4 ② 5 ③ 6 ④ 7

12 다음 중 실수 지수표기법(소문자) 서식 문자를 고르시오.

① %e ② %ld ③ %f ④ %F

13 다음 중 printf()에 사용하는 플래그 중 좌측 정렬에 사용하는 것을 고르시오.

① - ② 0 ③ + ④ #

14 다음 중 printf()에 사용하는 플래그 중 숫자 출력 시 +와 - 기호를 붙이는 것을 고르시오.

① - ② 0 ③ + ④ #

15 다음 중 printf()에 사용하는 플래그 중 숫자 앞 빈칸을 0으로 채우는 것을 고르시오.

① - ② 0 ③ + ④ #

16 다음 중 printf()에 사용하는 플래그 중 8진수 앞에 0을 붙이는 것을 고르시오.

① - ② 0 ③ + ④ #

17 다음 중 printf()에 사용하는 플래그 중 16진수 앞에 0x를 붙이는 것을 고르시오.

① - ② 0 ③ + ④ #

18 printf(" ", 123)에서 전체 10칸에 다음과 같이 출력되게 하려한다. 큰따옴표 안에 들어갈 올바른 서식 문자와 플래그는 무엇인가?

① %10d ② %-10d ③ %010d ④ %+d

19 printf(" ", 123)에서 전체 10칸에 다음과 같이 출력되게 하려한다. 큰따옴표 안에 들어갈 올바른 서식 문자와 플래그는 무엇인가?

```
0 0 0 0 0 0 0 1 2 3
```

① %10d ② %-10d ③ %010d ④ %+d

20 printf(" ", 123)에서 전체 10칸에 다음과 같이 출력되게 하려한다. 큰따옴표 안에 들어갈 올바른 서식 문자와 플래그는 무엇인가?

```
1 2 3
```

① %10d ② %-10d ③ %010d ④ %+d

21 printf(" ", 123)에서 전체 10칸에 다음과 같이 출력되게 하려한다. 큰따옴표 안에 들어갈 올바른 서식 문자와 플래그는 무엇인가?

```
+ 1 2 3
```

① %10d ② %-10d ③ %010d ④ %+d

22 printf(" ", 0x12)에서 전체 10칸에 다음과 같이 출력되게 하려한다. 큰따옴표 안에 들어갈 올바른 서식 문자와 플래그는 무엇인가?

```
0 x 1 2
```

① %#x ② %10.3f ③ %-10.3f ④ %f

23 printf(" ", 1.23456789)에서 전체 10칸에 다음과 같이 출력되게 하려한다. 큰따옴표 안에 들어갈 올바른 서식 문자와 플래그는 무엇인가?

```
1 . 2 3 4 5 6 8
```

① %#xd ② %10.3f ③ %-10.3f ④ %f

24 printf(" ", 1.23456789)에서 전체 10칸에 다음과 같이 출력되게 하려한다. 큰따옴표 안에 들어갈 올바른 서식 문자와 플래그는 무엇인가?

```
1 . 2 3 5
```

① %#xd ② %10.3f ③ %-10.3f ④ %f

13.2 파일의 이해

1 파일 입출력 이해하기

- 데이터를 저장하는 가장 간단한 방법이 파일
- 기존의 함수들과 유사한 함수들을 사용 - printf()의 파일 버전인 fprintf(), scanf()의 파일 버전인 fscanf(),이외에 fgetchar()나 fgets()와 같은 함수들도 있음

- 파일을 사용하기 위한 준비단계를 만들어 주는 함수가 fopen() -> fd = fopen("파일이름")
- 열쇠를 반환하는 작업이 fclose()
- 파일에 대한 작업 순서를 요약하면 fopen() - read/write – fclose() 임

13.2 파일의 이해

2 파일 기술자

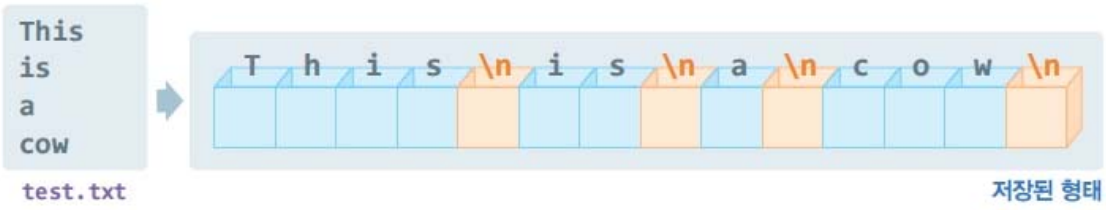
- fd = fopen("파일이름")으로 파일을 요청 했을 때 여러 검사에 통과하면 fd로 돌아오는 것이 파일에 접근할 수 있는 열쇠
- 열쇠의 정확한 이름은 파일 기술자(file descriptor)의 주소
- 파일 기술자의 주소 fd를 받으면 모든 작업은 파일 기술자로 이루어짐
- 파일 기술자에 있는 내용 중 가장 중요한 것이 파일 포인터 위치 지시자(file pointer) -> 현재 파일의 어떤 부분을 읽거나 쓰고 있는지를 기억
- 작업이 끝났다면 fclose(fd)를 사용하여 열었던 파일을 닫아 주어야 함

13.2 파일의 이해

3 파일의 구성

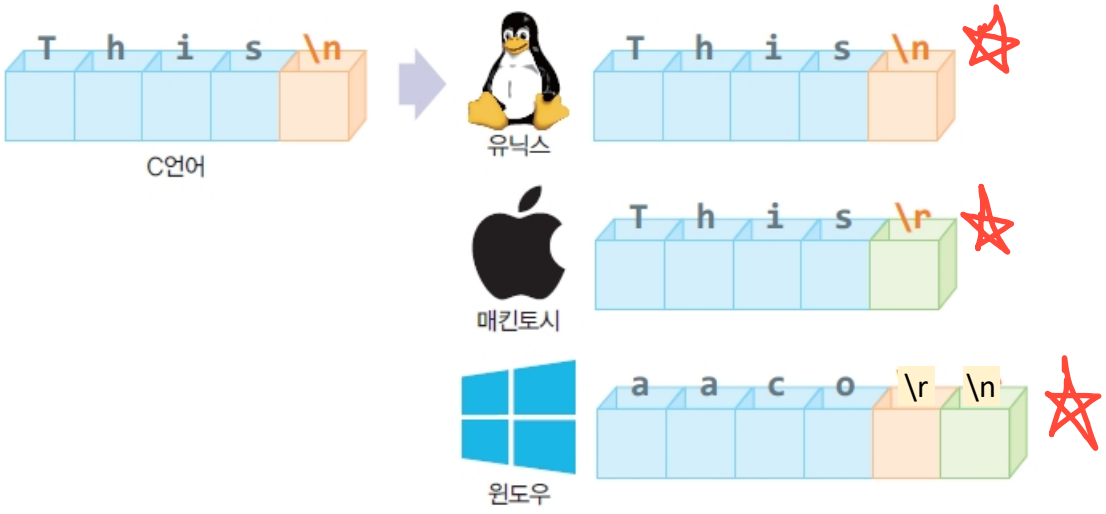
- 파일은 크게 텍스트 파일(text file)과 이진 파일(binary file)로 나뉨
- 데이터가 텍스트 형태(아스키 값)로 저장된 것이 텍스트 파일, 데이터가 이진수 형태로 저장 된 것이 이진 파일

- 주로 텍스트 파일로 작업 함 - 다음은 텍스트파일 저장 형태



13.2 파일의 이해

- C언어에서 한 줄의 끝을 알리는 줄바꿈 문자가 '₩n' 이지만, 저장 될 때에는 운영체제마다 다름
  - 유닉스/리눅스의 경우에는 C언어와 똑같이 줄의 끝을 알리는 문자는 '₩n'
  - 매킨토시의 경우에는 '₩n' 대신 CR(carriage return)문자인 '₩r'을 사용
  - 윈도우의 경우에는 두 개의 문자를 연달아 사용하여 "₩r₩n"



13.2 파일의 이해

4 파일 열고 닫기

- 파일에 접근하는 방법은 저수준과 고수준으로 나뉨
- open(), read(), write(), close() 저수준 함수 - 저수준은 불편하여 잘 사용 안함
- 고수준 파일 함수는 기존의 함수이름 앞에 f만 추가된 형태
  - open() - fopen(), close() - fclose(), read() – fread(). write() – fwrite(), printf() - fprintf(), scanf() - fscanf()와 같이 f로 시작하는 함수가 고수준 파일 관련 함수

- 파일을 여는 것은 fopen() 함수
  - 정상적으로 파일이 열렸을 경우, 파일 기술자의 포인터를 fp에게 줌
  - fp는 파일에 접근 할 수 있는 열쇠이다.
  - fp를 받으면, 파일을 읽고, 쓰고, 닫을 때에는 모두 fp를 통해서 이루어짐


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


13.2 파일의 이해

- fp는 FILE 구조체 포인터로 정의해서 사용
- 만약 파일이 없거나 여는데 문제가 있는 경우에는 널(NULL)이 반환됨
- fopen() 함수의 인자로는 "파일의 이름"과 "접근모드"가 사용
- 파일이름의 경우 작업하는 폴더(C 코드가 만들어지는 폴더)에 해당 파일이 있어야 함
- 접근 모드란 파일을 어떻게 사용할지를 결정하는 것이다. 기본적으로 읽기전용은 "r", 쓰기전용은 "w", 추가(append) 전용은 "a"

파일 열기 foepn()

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

(파일이름) (접근 모드)

13.2 파일의 이해

> 접근 모드와 특징

- 기존의 파일에 데이터가 있는 경우 "w"와 "w+"는 파일 안의 모든 내용을 지우기 때문에 조심해서 사용해야 함

표 13-6 접근 모드의 특징

모드	설명	파일이 있는 경우	파일이 없는 경우
"r"	읽기(read) 전용	정상	오류
"w"	쓰기(write) 전용	**내용물 삭제**	새 파일 만듦
"a"	추가(append) 전용	파일 끝에 이어 붙임	새 파일 만듦
"r+"	읽기 / 쓰기	정상	**오류**
"w+"	읽기 / 쓰기	**내용물 삭제**	새 파일 만듦
"a+"	읽기 / 추가	파일 끝에 이어 붙임	새 파일 만듦
t	텍스트 파일 - 단독 사용 불가	"rt" "wt" "at" "r+t" "w+t" "a+t"	
b	이진 파일 - 단독 사용 불가	"rb" "wb" "ab" "r+b" "w+b" "a+b"	

13.2 파일의 이해

- 파일을 다 사용하고 나면, 닫아주어야 함
- 파일을 닫는 fclose()의 형식은 다음과 같음.

파일 닫기 fclose()

```
fclose(fp);
```

중요

입출력의 끝과 파일의 끝은 EOF이다.

그러나, 파일 끝에 실제로 EOF 표시가 있는 것은 아니고 접근함수가 알아냄.

[예제 13-4] 파일 열고 닫기 코드

```c
#include <stdio.h>

int main() {
   FILE *fp = NULL;

   fp = fopen("test.txt", "r");
   if (fp == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   else printf("파일 open 성공\n");
   fclose(fp);

   return 0;
}
```

실행 화면

```
파일 open 성공
```


26 다음 중 파일에 접근하기 위한 준비 단계를 가리키는 단어는 무엇인지 고르시오.

① open ② close ③ read ④ write

27 다음 중 파일을 다 사용하고 열쇠를 반환하는 단계를 가리키는 단어는 무엇인지 고르시오.

① open ② close ③ read ④ write

28 다음 중 파일에 접근할 수 있는 열쇠의 정확한 이름을 가리키는 단어는 무엇인지 고르시오.

① 폴더 ② 스트림 ③ 버퍼 ④ 파일 기술자

30 다음 중 고수준 파일 관련 함수가 아닌 것은 무엇인지 고르시오.

① fopen() ② fclose() ③ read() ④ fwrite()

31 다음 파일 접근 모드 중 읽기 전용 접근 모드를 고르시오.

① w ② a ③ r ④ b

32 다음 파일 접근 모드 중 쓰기 전용 접근 모드를 고르시오.

① w ② a ③ r ④ b

33 다음 파일 접근 모드 중 추가 전용 접근 모드를 고르시오.

① w ② a ③ r ④ b

34 다음 파일 접근 모드 중 이진 파일 접근 모드를 고르시오.

① w ② a ③ r ④ b

35 다음 파일 접근 모드 파일이 없는 경우 오류가 나는 모드를 고르시오.

① w ② a ③ r ④ b

36 다음 파일 접근 모드 중 파일이 있는 경우 내용물이 삭제되는 모드를 고르시오.

① w ② a ③ r ④ b

37 다음 파일 접근 모드 중 파일이 있는 경우 파일 끝에 이어 붙이는 접근 모드를 고르시오.

① w ② a ③ r ④ b

38 다음 파일 접근 모드 중 단독으로 쓸 수 없는 접근 모드를 고르시오.

① w ② a ③ r ④ b

13.3 파일 입출력

1 문자 단위 입출력

- 기존의 문자 관련 함수들 앞에 f가 붙은 함수들이 파일 관련 입출력 함수
- 기존의 문자 관련 함수와 다른 점은 입출력이 모니터나 키보드에서 일어나는 것이 아니라 파일에서 발생
- fopen()을 사용하여 정상적으로 얻어진 파일 기술자 포인터 *fp를 사용하여 입출력
- 이진파일의 경우 입출력으로 오고가는 데이터가 문자가 아니기 때문에 기존의 문자관련 입출력 함수를 사용할 수 없음
  - 이진 데이터를 입출력하고 싶다면, fread()와 fwrite()를 사용

표 13-7 파일 입출력 관련 함수

종류		함수 정의
문자	입력	int fgetc(FILE *fp)
	출력	int fputc(int c, FILE *fp)
문자열	입력	char *fgets(char *buf, int n, FILE *fp)
	출력	char *fputs(const char *buf, FILE *fp)
서식	입력	int fscanf(FILE *fp, "" ...
	출력	int fprintf(FILE *fp, "" ...
이진	입력	size_t fread(char *buff, int size, int n, FILE *fp)
	출력	size_t fwrite(char *buff, int size, int n, FILE *fp)

13.3 파일 입출력

[예제 13-5] 문자 출력 코드

```c
#include <stdio.h>

int main() {
   FILE *fp = NULL;
   char buf[14] = "This is a cow";
   int k;

   fp = fopen("test.txt", "w");       // test.txt를 쓰기 전용으로 연다
   if (fp == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   printf("test.txt에 한 글자씩 쓰기\n");
   for (k = 0; k < 14; k++)
      fputc(buf[k], fp);              // 문자열의 내용을 한 글자씩 파일에 쓴다

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

실행 화면

```
test.txt에 한 글자씩 쓰기
```

test.txt 파일 내용

```
This is a cow
```

[예제 13-6] 문자 입력 코드

```c
#include <stdio.h>

int main() {
   FILE *fp = NULL;
   char c;

   fp = fopen("test.txt", "r");           // test.txt를 읽기 전용으로 연다
   if (fp == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   while ((c = fgetc(fp)) != EOF)         // 파일에서 한 글자씩 가져온다
      putchar(c);

   printf("\n");
   fclose(fp);
   return 0;
}
```

실행 화면

```
This is a cow
```

test.txt 파일 내용

```
This is a cow
```


[예제 13-7] 문자 복사 코드

```c
#include <stdio.h>

int main() {
   FILE *fpr = NULL, *fpw = NULL;
   char c;

   fpr = fopen("test.txt", "r");      // test.txt를 읽기 전용으로 연다
   fpw = fopen("test_b.txt", "w");    // test_b.txt를 쓰기 전용으로 연다
   if (fpr == NULL || fpw == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   printf("파일 복사 시작\n");
   while ((c = fgetc(fpr)) != EOF)    // EOF를 만날 때까지 한 글자씩 읽어온다
      fputc(c, fpw);                  // fpw가 가리키는 파일에 문자 c를 쓴다

   fclose(fpr);
   fclose(fpw);
   return 0;
}
```

실행 화면

```
파일 복사 시작
```

test.txt 파일 내용

```
This is a cow
Test is over
```

test_b.txt 파일 내용

```
This is a cow
Test is over
```

마무리

키워드로 끝내는 핵심 포인트

- **fopen** 함수가 파일을 개방하면 메모리에 스트림 파일(파일 구조체 + 버퍼)을 만든다.
- **스트림 파일**은 프로그램과 장치를 연결하며 버퍼에 데이터를 저장한다.
- 스트림 파일의 파일 구조체 주소를 파일 포인터로 받는다.
- **파일 입출력 함수**는 스트림 파일을 통해 입출력을 수행한다.
- **fclose** 함수는 개방한 스트림 파일을 메모리에서 제거한다.

마무리

표로 정리하는 핵심 포인트

표 18-1 파일 개방 함수와 문자 입출력 함수

구분	기능	사용 예
파일 열기 fopen	원형	`FILE *fopen(const char *, const char *);`
	사용 예	`FILE *fp;` `fp = fopen("a.txt", "r");`
	반환값	개방에 성공하면 FILE 포인터, 실패하면 널 포인터(NULL)
파일 닫기 fclose	원형	`int fclose(FILE *);`
	사용 예	`fclose(fp);`
	반환값	성공하면 0, 오류가 발생한 경우 EOF
문자 입력 fgetc	원형	`int fgetc(FILE *);`
	사용 예	`int ch;` `ch = fgetc(fp);`
	반환값	입력한 문자, 오류나 파일에 데이터가 없을 때 EOF
문자 출력 fputc	원형	`int fputc(int, FILE *);`
	사용 예	`fputc(ch, ofp);`
	반환값	출력한 문자, 오류가 발생한 경우 EOF

마무리

❖ 다음 중에서 스트림 파일에 포함되지 않는 것은?

① 스트림 버퍼
② 파일 구조체 주소
③ 버퍼의 입출력 위치 지시자
④ 파일 구조체 변수
⑤ EOF

마무리

❖ 다음 중에서 설명이 잘못된 것을 바로잡으시오.

① 파일을 개방하는 것은 입출력을 위한 준비 작업이다.
② 입출력 함수가 호출되면 각자 독립적인 스트림 버퍼를 만든다.
③ 하나의 프로그램은 입력 파일과 출력 파일을 각각 하나씩 개방할 수 있다.
④ 모든 파일에는 끝을 표시하는 EOF 문자가 있다.

13.3 파일 입출력

2 문자열 단위 입출력

```c
#include <stdio.h>

int main() {
   FILE *fpr = NULL, *fpw = NULL;
   char buf[80];

   fpr = fopen("test.txt", "r");          // test.txt를 읽기 전용으로 연다
   fpw = fopen("test_b.txt", "w");        // test_b.txt를 쓰기 전용으로 연다
   if (fpr == NULL || fpw == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   printf("파일 복사 시작\n");
   fgets(buf, 80, fpr);
   fputs(buf, fpw);

   fclose(fpr);
   fclose(fpw);
   return 0;
}
```

실행 화면

```
파일 복사 시작
```

test.txt 파일 내용

```
This is a cow
Test is over
```

test_b.txt 파일 내용

```
This is a cow
```


13.3 파일 입출력

중요

fgets()는 더 이상 읽을 것이 없을 때 반환하는 값이 EOF가 아닌 NULL이다.

fgets()는 줄바꿈문자('₩n')를 만나거나, EOF를 만나거나, 버퍼크기보다 한 개 작은 문자열까지만 입력받고 종료 된다. 맨 마지막에는 자동으로 NULL을 삽입해 준다.

```c
while (fgets(buf, 80, fpr) != NULL)
   fputs(buf, fpw);
```

파일 전체 복사 코드

```c
while (fgets(buf, 80, fpr) != EOF)
   fputs(buf, fpw);
```

NULL이 아닌 EOF는 무한루프에 빠짐

3 서식 문자를 사용한 입출력

```c
#include <stdio.h>

int main() {
   FILE *fp = NULL;

   fp = fopen("test.txt", "w");
   if (fp == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   printf("파일 쓰기 시작\n");
   fprintf(fp, "%s %d", "gildong", 1000);
   fprintf(fp, "%s %d", "hyungwook", 2000);

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

실행 화면

```
파일 쓰기 시작
```

test.txt 파일 내용

```
gildong 1000hyungwook 2000
```


[예제 13-10] 문자 열 복사 코드

```c
#include <stdio.h>

int main() {
   FILE *fp = NULL;
   char buf[80];
   int num;

   fp = fopen("test.txt", "r");
   if (fp == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   fscanf(fp, "%s %d", buf, &num);
   printf("%s %d\n", buf, num);
   fscanf(fp, "%s %d", buf, &num);
   printf("%s %d\n", buf, num);

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

실행 화면

```
gildong 1000
hyungwook 2000
```

test.txt 파일 내용

```
gildong 1000hyungwook 2000
```

파일의 끝을 알려 주는 함수

```
int feof(FILE *fp)
```

```
13  while (feof(fp) == 0){
14     fscanf(fp, "%s %d", buf, &num);
15     printf("%s %d\n", buf, num);
16  }
```

feof(fp)는 파일 끝을 알려 준다

4 이진 파일 입출력

- 파일에 이진수로 저장하거나 읽어오고 싶다면 파일을 열 때 "b"를 붙여서 연다
- 기존의 fprintf(), fscanf(), fgets(), fputs()와 같은 문자 입출력 함수를 사용할 수 없음
- 이진 형태의 입출력에 사용하는 함수는 fread()와 fwrite()
- fread()와 fwrite()는 블록(block)을 사용
  - fread(buf, 블록의 크기, 반복횟수, fp);의 형태
  - fwrite(buf, 블록의 크기, 반복횟수, fp);의 형태

fread()와 fwrite()

```
fread(buf, 블록의 크기, 반복 횟수, fp);
fwrite(buf, 블록의 크기, 반복 횟수, fp);
```

13.3 이진 파일 입출력

[예제 13-11] 이진 출력 코드

```c
#include <stdio.h>

typedef struct hero {                    // hero 구조체 정의
   char name[10];                        // hero 구조체 멤버: 이름(문자열)
   int age;                              // hero 구조체 멤버: 나이(정수)
   float power;                          // hero 구조체 멤버: 힘(실수)
} hero;                                  // hero 구조체 정의 끝

int main() {
   FILE *fp = NULL;
   hero buf[3] = { {"gildong", 22, 8.2},
                   {"hyungwook", 28, 9.3},
                   {"changsik", 21, 7.6} };

   fp = fopen("test.bin", "wb");       // test.bin을 이진 쓰기전용으로 연다
   if (fp == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   printf("test.bin 쓰기 시작\n");
   fwrite(buf, sizeof(hero), 3, fp);   // 구조체 hero 크기의 블록을 3개 쓴다

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

```
test.bin 쓰기 시작
```

test.bin 파일 내용

```
73 75 70 65 72 60 61 6E   00 00 CC CC 20 00 00 00
33 33 03 41 61 71 75 61   60 61 6E 00 00 00 CC CC
26 00 00 00 CD CC 14 41   73 70 69 64 65 72 6D 61
6E 00 CC CC 1F 00 00 00   33 33 F4 40
```


13.3 이진 파일 입출력

[예제 13-12] 이진 입력 코드

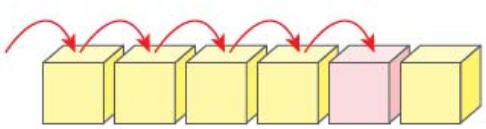
```c
#include <stdio.h>

typedef struct hero {                    // hero 구조체 정의
   char name[10];                        // hero 구조체 멤버: 이름(문자열)
   int age;                              // hero 구조체 멤버: 나이(정수)
   float power;                          // hero 구조체 멤버: 힘(실수)
} hero;                                  // hero 구조체 정의 끝

int main() {
   FILE *fp = NULL;
   hero buf[3] = { {"", } };           // hero 구조체 배열 초기화
   int k;

   fp = fopen("test.bin", "rb");       // test.bin을 이진 쓰기 전용으로 연다
   if (fp == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   fread(buf, sizeof(hero), 3, fp);    // 구조체 hero 크기의 블록을 3개 읽는다
   for (k = 0; k < 3; k++)
      printf("%s %d %.1f\n", buf[k].name, buf[k].age, buf[k].power);

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

실행 화면

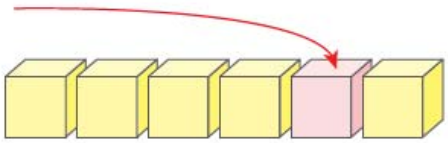
```
gildong 22 8.2
hyungwook 28 9.3
changsik 21 7.6
```

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

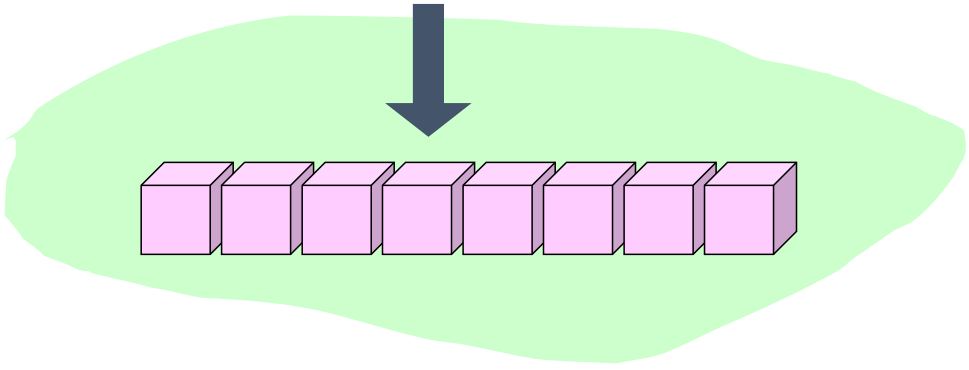


임의 접근파일

임의 접근 파일의 원리

- 파일(버퍼) 위치 지시자: 읽기와 쓰기 동작이 현재 어떤 위치에서 이루어지는 지를 나타낸다.

파일 위치 지시자



- 강제적으로 파일 위치 지시자를 이동시키면 임의 접근이 가능

fseek()

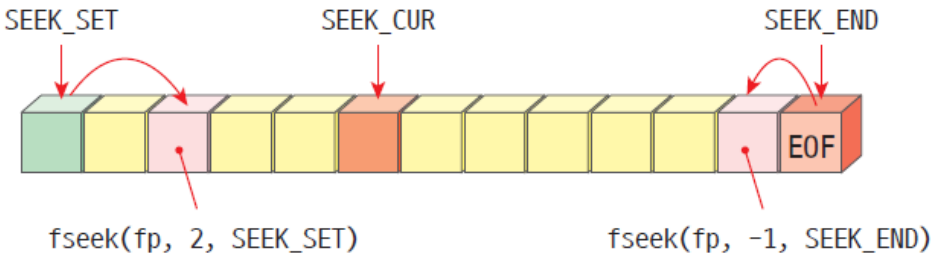
Syntax: fseek()

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

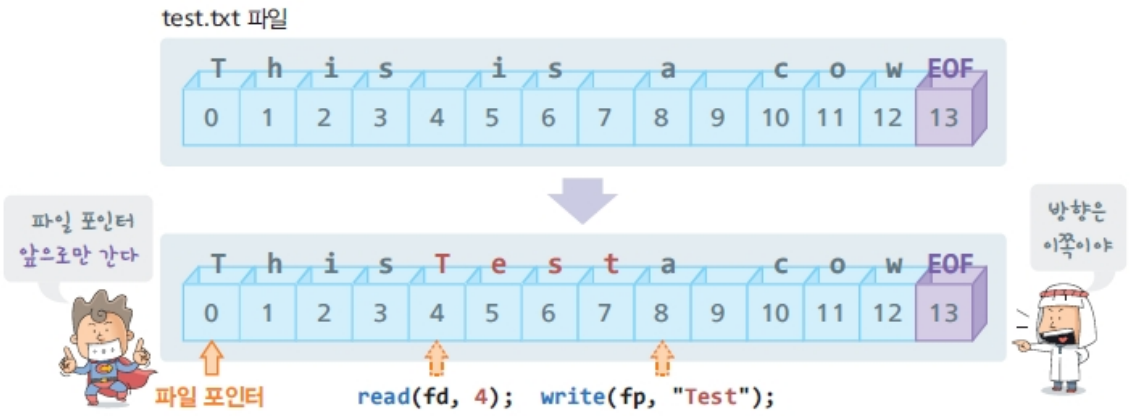
```
long ftell(FILE *fp);
```

Gets the current position of a file pointer.
The ftell function retrieves the current position of the file pointer (if any) associated with stream. The position is expressed as an offset relative to the beginning of the stream.

13.4 파일 포인터(위치 지시자)

1 파일 포인터(위치 지시자) 이해하기

- 파일 포인터는 파일을 사용할 수 있는 권한 뿐 아니라 현재 파일의 어느 위치에서 작업하고 있는지를 알려줌
- test.txt에 "This is a cow"가 들어 있고 fp = fopen("test.txt", "r")로 파일을 열었을 경우, 파일 포인터는 맨 앞의 'T'를 가리키고 있음
- 파일 포인터는 한 개만 존재 - read와 write 작업이 파일 포인터를 공유
- 파일 포인터는 어떤 작업을 하던 간에 앞으로 전진



13.4 파일 포인터

fseek() 파일 포인터 이동

```
fseek(FILE *fp, long offset, long mark);
```

- 파일 포인터를 이동시키는 함수가 fseek()
- 이동 거리(offset)는 얼마만큼 이동 시킬 것인지를 나타냄
- 어느 지점을 기준으로 이동할 지를 결정하는 것이 기준점(mark)
  - 기준점은 맨 앞은 SEEK_SET, 현재 위치는 SEEK_CUR, 맨 끝은 SEEK_END이다.

표 13-8 기준점 종류

기준점	숫자	비고
SEEK_SET	0	파일의 맨 앞 기준
SEEK_CUR	1	현재(current) 파일 포인터 위치 기준
SEEK_END	2	파일의 맨 끝 기준 - EOF의 위치 기준

13.4 파일 포인터

파일 포인터 위치 ftell()

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

2 파일 포인터 사용하기

- [예제 13-13]은 파일 포인터를 이동 시키는 함수 fseek()을 사용하여 파일 포인터의 특징을 보여주는 코드
- 7번 줄에서 "test.txt"를 "w+"모드로 열었음
- 12번 줄에서 fputs()를 사용하여 "0123456789"를 파일에 기록했다
- fputs() 이후의 파일의 데이터와 파일 포인터의 위치는 다음과 같음

[예제 13-13] 파일 포인터 이동 코드

```c
#include <stdio.h>

int main() {
   FILE *fp = NULL;
   long pos;

   fp = fopen("test.txt", "w+");       // test.bin을 이진 쓰기전용으로 연다
   if (fp == NULL) {
      printf("파일 open 실패\n");
      exit();
   }
   fputs("0123456789", fp);
   pos = ftell(fp);
   printf("현재 파일포인터 %d\n", pos);
   fseek(fp, 2, SEEK_SET);
   printf("2, SEEK_SET 값은 %c\n", fgetc(fp));
   fseek(fp, 3, SEEK_CUR);
   printf("3, SEEK_CUR 값은 %c\n", fgetc(fp));
   fseek(fp, -1, SEEK_END);
   printf("-1, SEEK_END 값은 %c\n", fgetc(fp));

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

실행 화면

```
현재 파일포인터 10
2, SEEK_SET 값은 2
3, SEEK_CUR 값은 6
-1, SEEK_END 값은 9
```

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;
}
```


■ 파일 위치 지시자 컨트롤의 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;
}
```

실행결과

```
836
```

■ 현재 파일 위치 지시자의 위치는? : ftell

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

파일 위치 지시자의 위치 정보

```
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);
        fseek(fp, -1, SEEK_END);
        putchar(fgetc(fp));

        fseek(fp, fpos, SEEK_SET);
    }
    fclose(fp);
    return 0;
}
```

실행결과

```
1-2-3-4-
```


■ 파일 위치 되돌리기: rewind

```
#include <stdio.h>
void rewind(FILE * stream);
```

반환형이 void 이므로 반환 값 없음

위의 함수 호출을 다음의 문장으로 대신할 수 있다!

fseek(fp, 0, SEEK_SET);

29 파일을 처음 열었을 경우 파일 포인터의 위치는 몇 번인가?

① -1 ② 0 ③ 1 ④ 2

39 fseek()에서 파일의 맨 앞 기준을 나타내는 단어를 고르시오.

① SEEK_STAR ② SEEK_CUR ③ SEEK_SET ④ SEEK_END

40 fseek()에서 현재 파일 포인터 기준을 나타내는 단어를 고르시오.

① SEEK_STAR ② SEEK_CUR ③ SEEK_SET ④ SEEK_END

41 fseek()에서 파일의 맨 끝 기준을 나타내는 단어를 고르시오.

① SEEK_STAR ② SEEK_CUR ③ SEEK_SET ④ SEEK_END

마무리

❖ 다음 파일 입출력 함수 중에서 반환값의 형태가 다른 함수는?

① fgetc
② fputc
③ fgets
④ fputs
⑤ fscanf
⑥ fprintf

마무리

실습

❖ 다음 프로그램의 실행결과를 적으시오.

```
FILE *fp;
char str[20] = “empty”
int ch;
fp = fopen(“a.txt”, “r”);
ch = fgetc(fp);
while (fgetc(fp) != EOF);
fgets(str, sizeof(str), fp);
printf(“%s”, str);
fclose(fp);
return 0;
```

a.txt의 내용:
a mango
an apple

문제 [문자열 방식의 데이터 입출력]

- 프로그램상에서 mystory.txt라는 이름의 파일을 만들어서 본인의 이름, 주민번호, 전화번호를 입력하는 프로그램을 작성하라. 입력의 형태는 다음과 같아야 한다.

  #이름: 홍길동
  #주민번호: 900209-1012589
  #전화번호: 010-1111-2222

  그리고 입력이 완성되면 메모장으로 확인이 가능해야 한다.

문제 [문자열 방식의 데이터 입출력]

- 앞 문제에서 작성한 파일에 데이터를 추가하자. 추가할 데이터는 즐겨 먹는 음식의 정보와 취미이다. 입력의 형태는 다음과 같아야 한다.

  #즐겨먹는 음식: 짜장면, 탕수육

  #취미: 축구

  그리고 입력이 완성되면 메모장으로 확인이 가능해야 한다.

마무리

단어 검출 프로그램

❖ 텍스트 파일에서 등록된 단어 이외의 단어를 찾아 새로운 파일에 출력하라. 모든 단어의 길이는 최대 20자, 등록 단어 수는 최대 10개로 제한하며 검출 대상 단어 수는 제한이 없다.

a.txt 등록 단어 파일
dog tiger horse monkey lion koala giraffe owl

b.txt 입력 단어 파일
lion elephant pear dog tiger apple kangraroo orange bear owl

c.txt 결과 파일
elephant pear apple kangaroo orange bear