

---

# 파일입출력

- Chapter 11 -

---

# 학습목차

---

- I. 파일에서 문자열 읽고 쓰기
- II. 파일 포인터 활용

# 파일에 문자열 쓰기 / 읽기

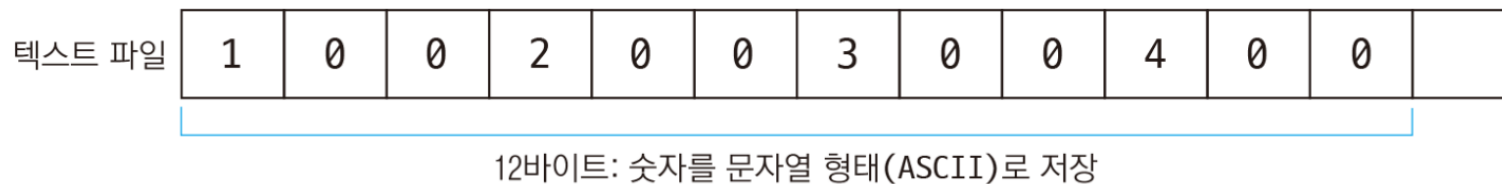
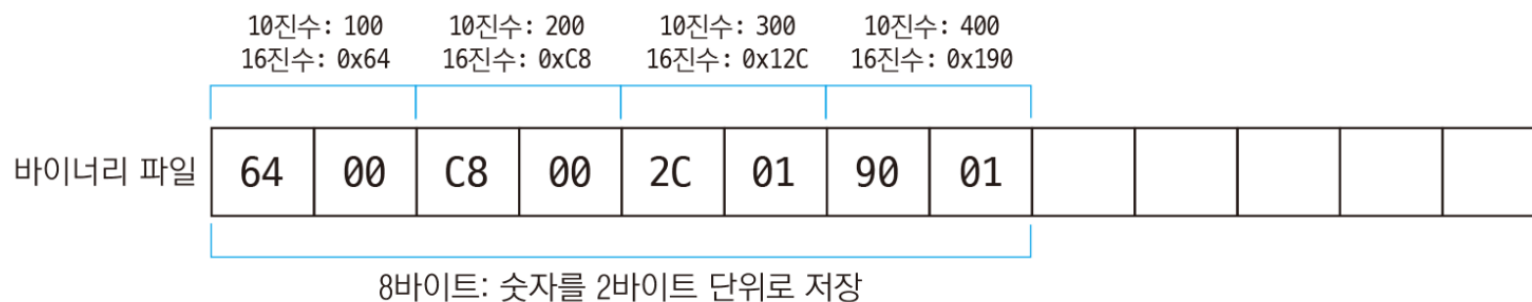
## ▶ 파일에 데이터 쓰기

### ▶ 파일처리는 프로그래밍에서 중요한 부분을 차지

▷ 지금까지 printf로 문자열을 화면에 출력하거나 sprintf 함수로 문자열을 생성

▷ 프로그래밍에 의해 처리된 결과를 영구적으로 저장 가능

▷ 문자나 바이너리 형식으로 저장



# 파일에 문자열 쓰기 / 읽기

## ▶ 문자열을 파일에 저장하는 방법

▶ `FILE *포인터이름 = fopen(파일명, 파일모드);`

▷ `FILE *fopen(char const * _FileName, char const * _Mode);`

▷ 성공하면 파일 포인터를 반환, 실패하면 NULL을 반환

▶ `fprintf(파일포인터, 서식, 값1, 값2, ...);`

▷ `int fprintf(FILE *const _Stream, char const* const _Format, ...);`

▷ 성공하면 쓴 문자열의 길이를 반환, 실패하면 음수를 반환

▶ `fclose(파일포인터);`

▷ `int fclose(FILE *_stream);`

▷ 성공하면 0을 반환, 실패하면 EOF(-1)를 반환

`fopen()`



`fprintf()`

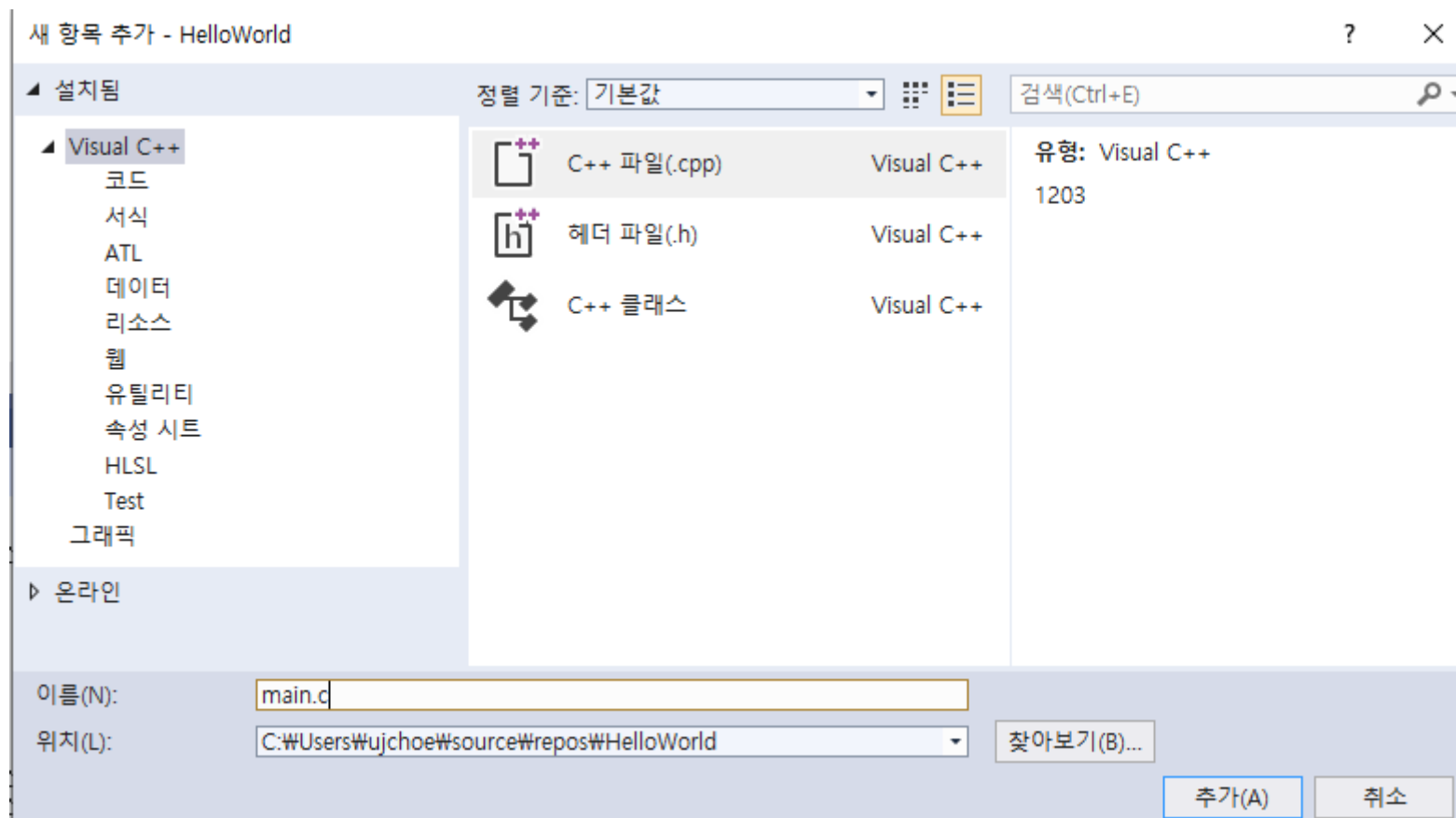


`fclose()`

# 파일에 문자열 쓰기 / 읽기

## ▶ 문자열 파일 예제

- ▶ 파일의 위치를 지정하지 않는 경우에는 기본적으로 .c 파일이 있는 폴더에 파일이 생성됨
- ▶ 소스코드를 추가하는 과정에서 저장 위치 확인



# 파일에 문자열 쓰기 / 읽기

## ▶ 문자열 파일 예제

▶ fopen() → fprintf() → fclose()

```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                // fopen, fprintf, fclose 함수가 선언된 헤더 파일

int main(){
    FILE *fp = fopen("hello.txt", "w");    // hello.txt 파일을 쓰기 모드(w)로 열기.
                                           // 파일 포인터를 반환
    fprintf(fp, "%s %d\n", "Hello", 100);  // 서식을 지정하여 문자열을 파일에 저장

    fclose(fp);    // 파일 포인터 닫기

    return 0;
}
```

Microsoft Visual Studio 디버그 콘솔

C:\Users\Wujchoe\source\repos\HelloWorld\Debug\HelloWorld.exe(프로세스 2712개)이(가) 종료되었습니다(코드: 0개).  
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용  
하도록 설정합니다.  
이 창을 닫으려면 아무 키나 누르세요...

# 파일에 문자열 쓰기 / 읽기

## ▶ 문자열 파일 예제

### ▶ 확인한 소스파일 저장 위치에 생성된 .txt파일 확인

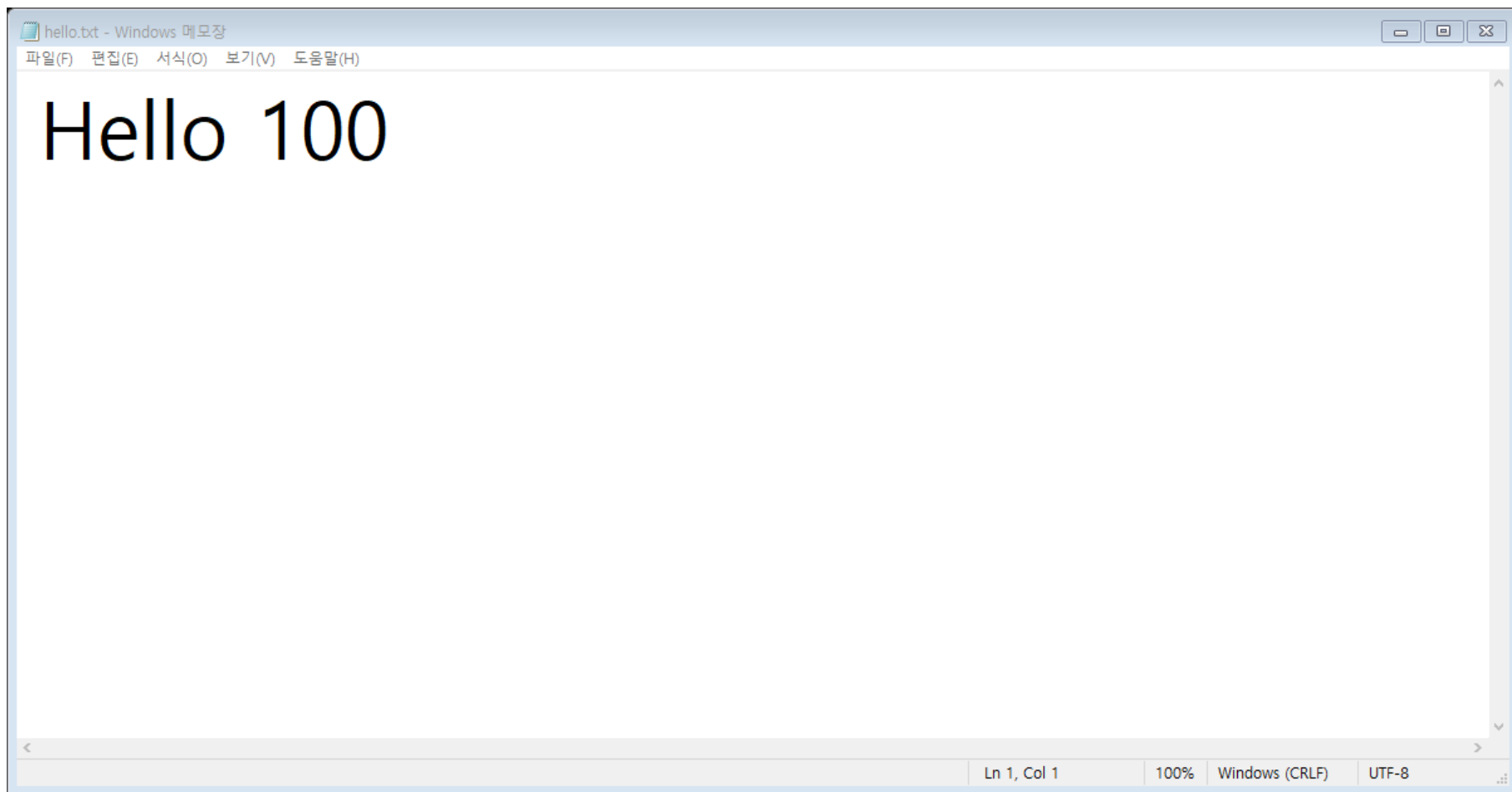
ujchoe > source > repos > HelloWorld

이름	수정한 날짜	유형	크기
.vs	2020-03-20 오전 4:09	파일 폴더	
Debug	2020-06-12 오전 4:13	파일 폴더	
Release	2020-06-10 오후 8:10	파일 폴더	
x64	2020-03-26 오후 4:46	파일 폴더	
hello.txt	2020-06-12 오전 4:13	텍스트 문서	1KB
HelloWorld.sln	2020-03-20 오전 4:10	Visual Studio Sol...	2KB
HelloWorld.vcxproj	2020-06-12 오전 4:13	VC++ Project	7KB
HelloWorld.vcxproj.filters	2020-06-12 오전 4:13	VC++ Project Filt...	1KB
HelloWorld.vcxproj.user	2020-03-20 오전 4:09	Per-User Project ...	1KB
main.c	2020-06-12 오전 4:13	C Source	1KB
practice_03.c	2020-04-10 오전 3:40	C Source	1KB
test1.c	2020-04-10 오전 2:48	C Source	2KB

# 파일에 문자열 쓰기 / 읽기

## ▶ 문자열 파일 예제

### ▶ 확인한 소스파일 저장 위치에 생성된 .txt파일 확인





# 파일에 문자열 쓰기 / 읽기

## ▶ 파일 열기

- ▶ 파일을 사용하기 위해서는 fopen 함수로 파일 포인터를 얻어야 함
- ▶ 아래의 경우 첫번째 인자는 파일명이며 두번째 인자인 w는 write를 의미하여 해당 파일이 있으면 덮어쓰고 없으면 새로 생성
- ▶ FILE은 stdio.h에 정의된 구조체이며 FILE과 \*를 합쳐서 파일 포인터라고 지칭

파일 포인터                      파일 이름 또는 파일 경로

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

파일 모드

- ▶ 파일 경로가 프로젝트마다 다르므로 동일한 위치에 저장할 경우 경로를 포함
  - ▶ 절대경로 : `fopen("C:\Users\Wujchoe\Documents\output\hello.txt", "w");`
  - ▶ 상대경로 : `fopen("../hello.txt", "w");`

# 파일에 문자열 쓰기 / 읽기

## ▶ 파일 모드

▶ 파일 열기에 성공하면 파일 포인터를 반환하고 실패하면 NULL을 반환

▶ 파일 모드는 일반적으로 rb, rt, w+b, w+t와 같이 읽기/쓰기 모드와 텍스트/바이너리 모드로 조합하여 사용

파일 모드	기능	설명
"r"	읽기 전용	파일을 읽기 전용으로 열기 / 단, 파일이 반드시 존재해야 함
"w"	쓰기 전용	새 파일을 생성 / 만약 파일이 있으면 새로운 내용으로 덮어씀
"a"	추가	파일을 열어 기존 데이터의 뒤에서부터 이어서 쓰기 / 만약 파일이 없으면 파일을 새로운 파일을 생성
"r+"	읽기/쓰기	파일을 읽기/쓰기용으로 열기 / 단, 파일이 반드시 있어야 하며 파일이 없으면 NULL을 반환
"w+"	읽기/쓰기	파일을 읽기/쓰기용으로 열기 / 파일이 없으면 파일을 생성하고, 파일이 있으면 내용을 덮어씀
"a+"	추가 (읽기/쓰기)	파일을 열어 파일 끝에 값이 추가 / 만약 파일이 없으면 파일을 생성 읽기는 파일의 모든 구간에서 가능하지만, 쓰기는 파일의 끝에서만 가능
t	텍스트 모드	문자로 읽고 쓰기
b	바이너리 모드	파일의 내용을 바이너리 형식으로 읽고 쓰기

# 파일에 문자열 쓰기 / 읽기

## ▶ 파일 쓰기

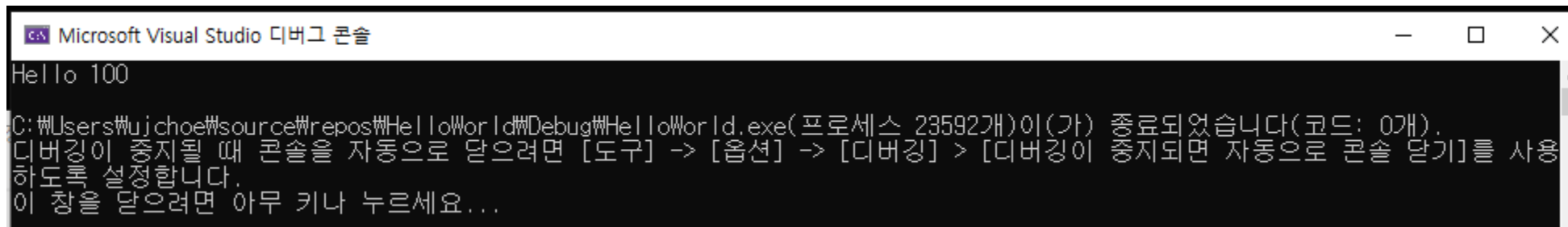
- ▶ fprintf함수로 문자열을 파일에 쓰기

- ▶ fprintf(파일 포인터, 서식, 값1, 값2, ...)

  - ▷ fprintf(fp, "%s %d", "Hello", 100);

- ▶ stdout 매크로를 사용하면 문자열을 화면에 출력가능

  - ▷ fprintf(stdout, "%s %d\\n", "Hello", 100);



```
Microsoft Visual Studio 디버그 콘솔
Hello 100
C:\Users\hujchoe\source\repos\HelloWorld\Debug\HelloWorld.exe(프로세스 23592개)이(가) 종료되었습니다(코드: 0개).
디버깅이 중지될 때 콘솔을 자동으로 닫으려면 [도구] -> [옵션] -> [디버깅] > [디버깅이 중지되면 자동으로 콘솔 닫기]를 사용
하도록 설정합니다.
이 창을 닫으려면 아무 키나 누르세요...
```

# 파일에 문자열 쓰기 / 읽기

## ▶ 파일 닫기

- ▶ 파일 쓰기가 종료되면 반드시 `fclose()` 함수로 파일 포인터를 닫아줌

  - ▶ `fclose(fp)`

- ▶ 파일 포인터도 구조체 `FILE` 크기 만큼 동적 메모리를 할당한 것으로 사용 후 닫지 않으면 메모리 누수현상 발생

# 파일에 문자열 쓰기 / 읽기

## ▶ 서식을 지정하여 파일에서 문자열 읽기

- ▶ `fscanf(파일포인터, 서식, 변수의 주소1, 변수의 주소2, ...);`
- ▶ `int fscanf(FILE *const _Stream, char const *const _Format, ...);`
  - ▶ 성공하면 읽어온 값의 개수를 반환, 실패하면 EOF(-1)를 반환
- ▶ 파일 쓰기 과정과 동일

`fopen()`



`fscanf()`



`fclose()`

# 파일에 문자열 쓰기 / 읽기

▶ 서식을 지정하여 파일에서 문자열 읽기

▶ fscanf함수도 stdin 매크로를 이용하여 사용자 입력한 값을 변수에 저장 가능

```
char s1[10];  
int num1;  
fscanf(stdin, "%s %d", s1, &num1);    // 서식을 지정하여 표준 입력(stdin)에서 문자열 읽기
```

# 파일에 문자열 쓰기 / 읽기

## ▶ 파일 스트림

- ▶ fprintf, fscanf 등의 함수의 매개변수에서 파일 포인터 부분을 보면 FILE\* const \_Stream와 같이 스트림(stream)이라고 되어 있음
- ▶ 보통 파일 포인터를 파일 스트림이라고도 하는데 스트림은 물 등의 액체가 흐르는 것을 의미
- ▶ 파이프 속에 물이 계속 흘러다니는 것처럼 파일 스트림도 파일의 데이터를 연속적으로 처리한다고 해서 스트림
  - ▶ 즉, 파일에서 데이터를 처리할 때마다 매번 파일을 여는 것이 아니라 파일 스트림을 한 번 생성해서 계속 데이터를 쓰거나 가져오는 방식
  - ▶ fopen으로 파일을 읽기 전용으로 열면 입력 스트림 쓰기
  - ▶ 전용으로 열면 출력 스트림, 읽기/쓰기로 열면 입출력 스트림
  - ▶ stdin은 입력 스트림, stdout, stderr는 출력 스트림

# 파일에 문자열 쓰기 / 읽기

▶ 서식을 지정하여 파일에서 문자열 읽기

▶ 앞서 저장한 hello.txt 파일에서 문자열 읽기

```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                // fopen, fscanf, fclose 함수가 선언된 헤더 파일

int main(){
    char s1[10];
    int num1;

    FILE *fp = fopen("hello.txt", "r");    // hello.txt 파일을 읽기 모드(r)로 열기.
                                           // 파일 포인터를 반환
    fscanf(fp, "%s %d", s1, &num1);    // 서식을 지정하여 파일에서 문자열 읽기

    printf("%s %d\n", s1, num1);        // Hello 100: 파일에서 읽은 값을 출력

    fclose(fp);    // 파일 포인터 닫기

    return 0;
}
```

Hello 100



# 파일에 문자열 쓰기 / 읽기

## ▶ 서식 없이 파일에 문자열 쓰기

### ▶ fputs(버퍼, 파일포인터);

▷ fprintf와 달리 데이터의 서식을 지정하지 않음

▷ int fputs(char const \*\_Buffer, FILE \*\_Stream);

▷ 성공하면 음수가 아닌 값을 반환, 실패하면 EOF(-1)을 반환

▷ fputs 함수도 파일 포인터 대신 stdout을 지정하면 문자열이 표준 출력(화면)에 출력

```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                // fopen, fputs, fclose 함수가 선언된 헤더 파일

int main(){
    FILE *fp = fopen("hello.txt", "w");    // hello.txt 파일을 쓰기 모드(w)로 열기.
                                           // 파일 포인터를 반환
    fputs("Hello, world!", fp);    // 파일에 문자열 저장
    fclose(fp);                    // 파일 포인터 닫기
    return 0;
}
```

fopen()



fputs()



fclose()

Hello, world!

# 파일에 문자열 쓰기/읽기

## ▶ 파일에 문자열 쓰기

▶ fwrite(버퍼, 쓰기크기, 쓰기횟수, 파일포인터);

▶ `size_t fwrite(void const *_Buffer, size_t _ElementSize, size_t _ElementCount, FILE *_Stream);`

▶ fwrite 함수는 fputs 함수와는 달리 쓰기 크기와 쓰기 횟수를 지정

▶ 성공한 쓰기 횟수를 반환, 실패하면 지정된 쓰기 횟수보다 작은 값을 반환

▶ fwrite 함수도 파일 포인터 대신 stdout을 지정하면 문자열이 화면에 출력

```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                // fopen, fputs, fclose 함수가 선언된 헤더 파일

int main(){
    FILE *fp = fopen("hello.txt", "w");    // hello.txt 파일을 쓰기 모드(w)로 열기.
                                           // 파일 포인터를 반환
    fputs("Hello, world!", fp);    // 파일에 문자열 저장

    fclose(fp);    // 파일 포인터 닫기

    return 0;
}
```

fopen()



fwrite()



fclose()

# 파일에 문자열 쓰기 / 읽기

## ▶ 파일에서 문자열 읽기

▶ fgets(버퍼, 버퍼크기, 파일포인터);

▶ char \*fgets(char \*\_Buffer, int \_MaxCount, FILE \*\_Stream);

▶ 성공하면 읽은 문자열의 포인터를 반환, 실패하면 NULL을 반환

▶ 입력된 버퍼 사이즈만큼만 읽어옴

▶ 아래코드에서는 20바이트를 입력하였는데 마지막은 공백 문자가 포함되므로 실제로 19바이트만 읽어 들임

```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                // fopen, fgets, fclose 함수가 선언된 헤더 파일
int main(){
    char buffer[20];              // 파일을 읽을 때 사용할 임시 공간

    FILE *fp = fopen("hello.txt", "r");    // hello.txt 파일을 읽기 모드로 열기.
                                           // 파일 포인터를 반환
    fgets(buffer, sizeof(buffer), fp);    // hello.txt에서 문자열을 읽음
    printf("%s\n", buffer);              // Hello, world!: 파일의 내용 출력
    fclose(fp);                          // 파일 포인터 닫기
    return 0;
}
```

fopen()



fgets()



fclose()

# 파일에 문자열 쓰기 / 읽기

## ▶ 파일에서 문자열 읽기

▶ fread(버퍼, 읽기크기, 읽기횟수, 파일포인터);

▶ size\_t fread(void \*\_Buffer, size\_t \_ElementSize, size\_t \_ElementCount, FILE \*\_Stream);

▶ 성공한 읽기 횟수를 반환, 실패하면 지정된 읽기 횟수보다 작은 값을 반환

▶ fread 함수를 사용할 때는 char 배열을 선언한 뒤 반드시 0으로 초기화

▶ 만약 앞에서 buffer를 0(NULL)으로 초기화하지 않고 fread로 파일을 읽으면 "Hello, world!"이외에도 쓸데없는 값들이 함께 출력

```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                // fopen, fread, fclose 함수가 선언된 헤더 파일
int main(){
    char buffer[20] = { 0, };    // 파일을 읽을 때 사용할 임시 공간, 미리 0으로 전부 초기화
    FILE *fp = fopen("hello.txt", "r");    // hello.txt 파일을 읽기 모드로 열기.
                                           // 파일 포인터를 반환
    fread(buffer, sizeof(buffer), 1, fp);    // hello.txt에서 버퍼 크기(20바이트)만큼 1번 값을 읽음
    printf("%s\n", buffer);    // Hello, world!: 파일의 내용 출력
    fclose(fp);    // 파일 포인터 닫기
    return 0;
}
```

fopen()



fread()



fclose()

# 파일에 문자열 쓰기 / 읽기

## ▶ buffer 초기화 여부에 따른 차이



# 퀴즈 01

▶ 다음 중 파일을 여는 방법으로 올바른 것을 고르세요.

1. `FILE *fp = fopen("hello.txt");`
2. `FILE *fp = fopen("hello.txt", "b");`
3. `FILE *fp = fopen("hello.txt", "r");`
4. `FILE *fp = fopen("hello.txt", "r", "w", "b");`
5. `FILE *fp = fopen("w", "hello.txt");`

## 퀴즈 02

▶ 다음 중 fputs의 매개변수 설명으로 올바른 것을 고르세요.

1. fputs(버퍼);
2. fputs(파일포인터, 버퍼);
3. fputs(버퍼, 버퍼크기, 파일포인터);
4. fputs(버퍼, 파일포인터);
5. fputs(파일포인터);

## 퀴즈 03

▶ 다음 중 fgets 함수로 문자열을 읽는 방법으로 올바른 것을 고르세요. 버퍼는 `char buffer[10]`, 파일 포인터는 `fp`로 선언되어 있습니다.

1. `fgets(buffer, sizeof(buffer), fp);`
2. `fgets(buffer, fp);`
3. `fgets(buffer, 0, fp);`
4. `fgets(fp, buffer);`
5. `fgets(fp, sizeof(buffer));`



## 퀴즈 04

▶ 다음 중 fwrite 매개변수에 대한 설명으로 올바른 것을 고르세요.

1. fwrite(버퍼, 쓰기횟수, 쓰기크기, 파일포인터);
2. fwrite(쓰기횟수, 쓰기크기, 버퍼, 파일포인터);
3. fwrite(쓰기횟수, 버퍼, 쓰기크기, 파일포인터);
4. fwrite(버퍼, 쓰기크기, 쓰기횟수, 파일포인터);
5. fwrite(파일포인터, 버퍼, 쓰기크기, 쓰기횟수);

## 퀴즈 05

▶ 다음 중 fread함수로 10바이트만큼 읽는 방법으로 올바른 것을 고르세요. 버퍼는 buffer, 파일 포인터는 fp로 선언되어 있습니다.

1. fread(buffer, 2, 5, fp);
2. fread(buffer, 1, 8, fp);
3. fread(buffer, 3, 3, fp);
4. fread(buffer, 10, 2, fp);
5. fread(buffer, 7, 6, fp);

# 퀴즈 05

▶ 다음 중 파일 모드의 설명으로 올바른 것을 모두 고르세요.

1. "r": 파일을 읽기 전용으로 엽니다. 만약 파일이 없을 경우 파일을 생성합니다.
2. "w": 값을 쓸 수 있도록 새 파일을 생성합니다. 만약 파일이 있을 경우 내용을 덮어씁니다.
3. "a": 파일의 처음에 값을 추가하도록 파일을 엽니다. 만약 파일이 없을 경우 파일을 생성합니다.
4. "t": 바이너리로 처리합니다.
5. "r+": 파일의 내용을 읽거나 쓸 수 있도록 파일을 엽니다. 단, 파일이 있어야 합니다.

## 퀴즈 07

▶ 다음 중 파일에 서식을 지정하여 문자열을 읽고 쓰는 방법으로 올바른 것을 모두 고르세요. 변수는 int num1, float num2, 파일 포인터는 fp로 선언되어 있습니다.

1. `fprintf("%d %f", num1, num2);`
2. `fprintf(fp, "%d %f", num1, num2);`
3. `fscanf("%d %f", num1, num2);`
4. `fscanf(fp, "%d %f", num1, num2);`
5. `fscanf(fp, "%d %f", &num1, &num2);`

# 실습문제 01

▶ 다음 소스 코드를 완성하여 문자열을 hello.txt 파일로 저장하시오.

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

int main()
{
    char s1[20] = "안녕하세요.";

    FILE *fp = ①_____

    ②_____

    fclose(fp);

    return 0;
}
```

## 실습문제 02

▶ position.txt에는 "x 30 y 20" 저장되어 있습니다. 다음 소스 코드를 완성하여 30 20을 출력하세요

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

int main()
{
    char c1, c2;
    int x, y;

    FILE *fp = ①_____

    ②_____

    printf("%d %d\n", x, y);

    fclose(fp);

    return 0;
}
```

30 20

# 파일 포인터 활용하기

## ▶ 파일 크기 구하기

- ▶ 파일을 다룰 정도의 크기로 버퍼를 설정하여 메모리의 낭비를 줄일 필요가 있음

- ▶ `fseek(파일포인터, 이동할 크기, 기준점);`

  - ▷ `int fseek(FILE *_Stream, long _Offset, int _Origin);`

  - ▷ 성공하면 0, 실패하면 -1을 반환

- ▶ `ftell(파일포인터);`

  - ▷ `long ftell(FILE *_Stream);`

  - ▷ 파일 포인터의 현재 위치를 반환, 실패하면 -1을 반환

# 파일 포인터 활용하기

## ▶ 파일 크기 구하기

```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                // fopen, fseek, ftell, fclose 함수가 선언된 헤더 파일

int main()
{
    int size;

    FILE *fp = fopen("hello.txt", "r");    // hello.txt 파일을 읽기 모드(r)로 열기.
                                           // 파일 포인터를 반환

    fseek(fp, 0, SEEK_END);              // 파일 포인터를 파일의 끝으로 이동시킴
    size = ftell(fp);                    // 파일 포인터의 현재 위치를 얻음

    printf("%d\n", size);                // 13

    fclose(fp);

    return 0;
}
```

13



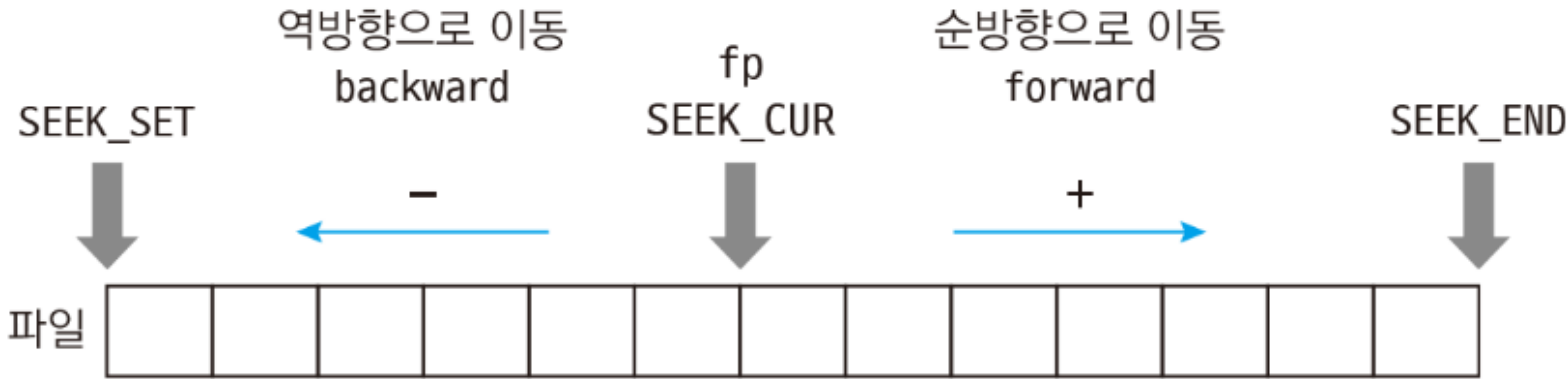
# 파일 포인터 활용하기

## ▶ 파일 크기 구하기

### ▶ fseek 함수의 기준점 종류

기준점	설명	예
SEEK_SET	파일의 처음부터 이동을 시작	fseek(fp, 0, SEEK_SET); // 파일 포인터를 파일의 처음으로 이동시킴
SEEK_CUR	현재 위치부터 이동을 시작	fseek(fp, -10, SEEK_CUR); // 파일 포인터를 현재 위치에서 10바이트만큼 역방향으로 이동시킴(-10이 음수이므로)
SEEK_END	파일의 끝부터 이동을 시작	fseek(fp, 0, SEEK_END); // 파일 포인터를 파일의 끝으로 이동시킴

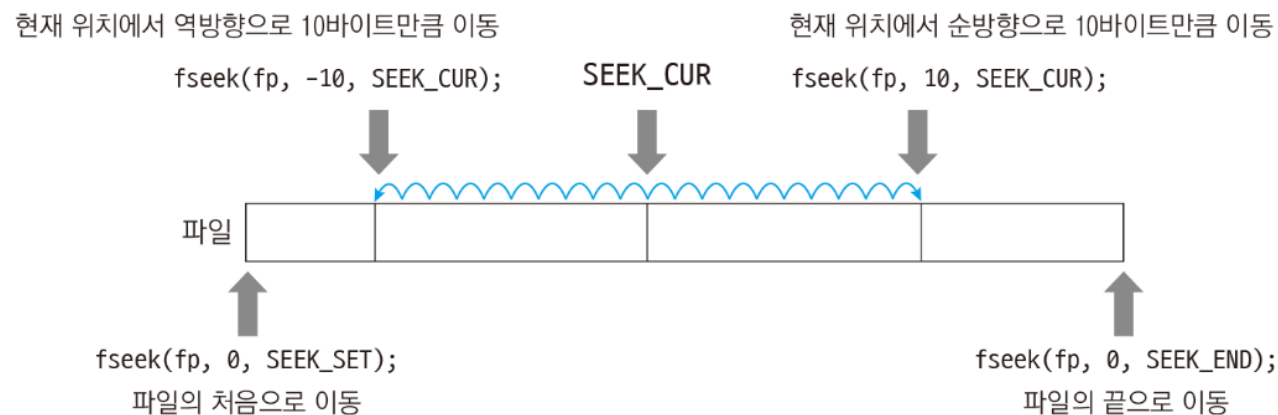
### ▶ 반환 값의 저장 과정



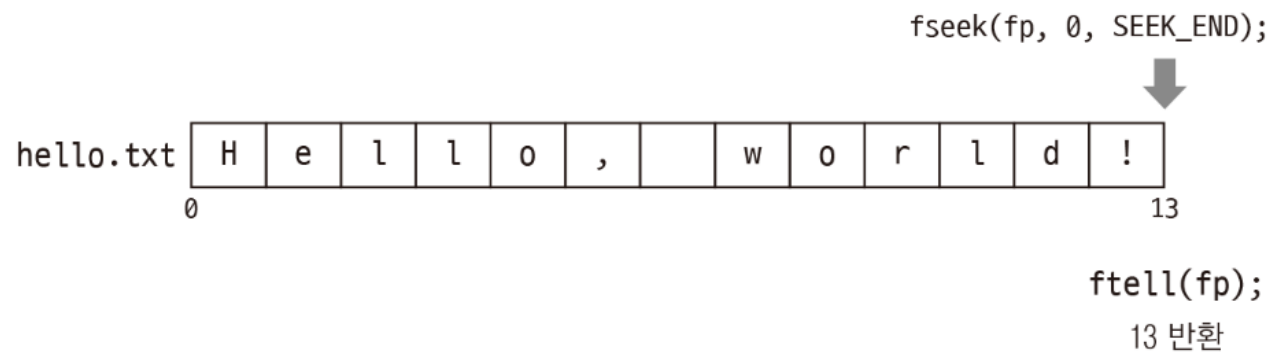
# 파일 포인터 활용하기

## ▶ 파일 크기 구하기

### ▶ fseek 함수의 사용



### ▶ fseek, ftell 함수로 파일 크기 구하기



# 파일 포인터 활용하기

## ▶ 파일 크기만큼 파일 읽기

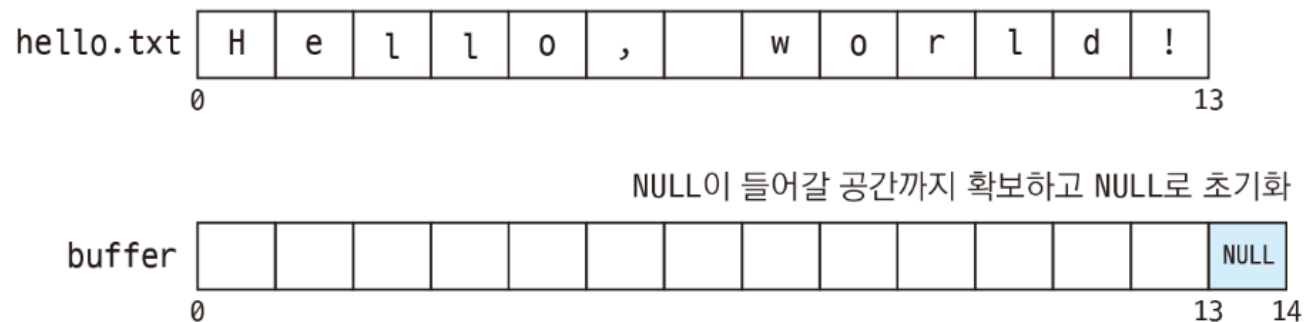
```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                 // fopen, fseek, ftell, fread, fclose 함수가 선언된 헤더 파일
#include <stdlib.h>                // malloc, free 함수가 선언된 헤더 파일
#include <string.h>                // memset 함수가 선언된 헤더 파일
int main(){
    char *buffer;
    int size;
    int count;
    FILE *fp = fopen("hello.txt", "r");    // hello.txt 파일을 읽기 모드(r)로 열기.
                                           // 파일 포인터를 반환
    fseek(fp, 0, SEEK_END);              // 파일 포인터를 파일의 끝으로 이동시킴
    size = ftell(fp);                    // 파일 포인터의 현재 위치를 얻음
    buffer = malloc(size + 1);           // 파일 크기 + 1바이트(문자열 마지막의 NULL)만큼 동적 메모리 할당
    memset(buffer, 0, size + 1);         // 파일 크기 + 1바이트만큼 메모리를 0으로 초기화
    fseek(fp, 0, SEEK_SET);               // 파일 포인터를 파일의 처음으로 이동시킴
    count = fread(buffer, size, 1, fp);   // hello.txt에서 파일 크기만큼 값을 읽음
    printf("%s size: %d, count: %d\n", buffer, size, count);
                                           // Hello world! size: 13, count: 1: 파일의 내용, 파일 크기, 읽은 횟수 출력
    fclose(fp);                          // 파일 포인터 닫기
    free(buffer);                         // 동적 메모리 해제
    return 0;
}
```

Hello, world! size: 13, count: 1

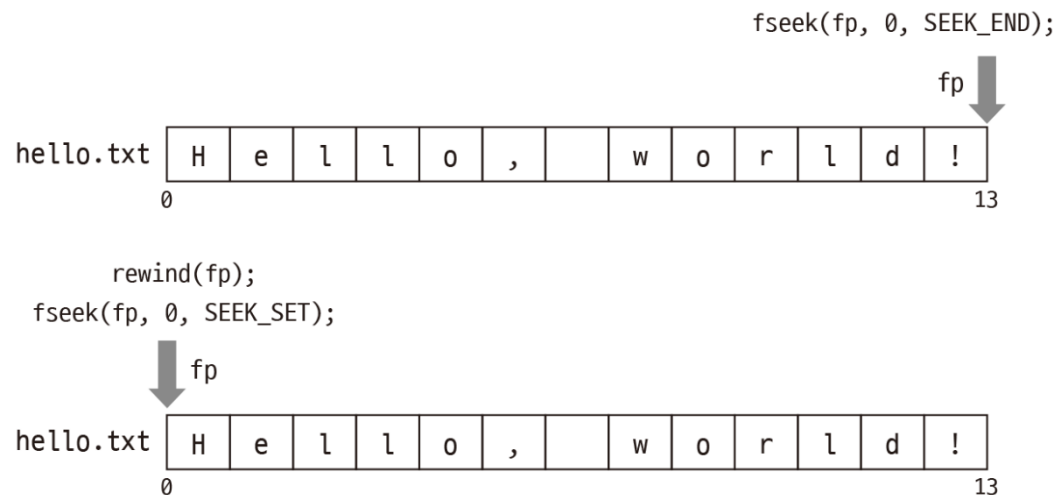
# 파일 포인터 활용하기

## ▶ 파일 크기만큼 파일 읽기

### ▶ 버퍼를 생성할 때 NULL이 들어갈 공간까지 확보하고 NULL로 초기화



### ▶ 파일 포인터를 파일 처음, 끝으로 이동시키기



# 파일 포인터 활용하기

## ▶ 파일을 부분적으로 읽고 쓰기

```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                 // fopen, fseek, fread, fclose 함수가 선언된 헤더 파일
#include <string.h>                // memset 함수가 선언된 헤더 파일
int main(){
    char buffer[10] = { 0, };
    FILE *fp = fopen("hello.txt", "r");    // hello.txt 파일을 읽기 모드(r)로 열기.
                                           // 파일 포인터를 반환

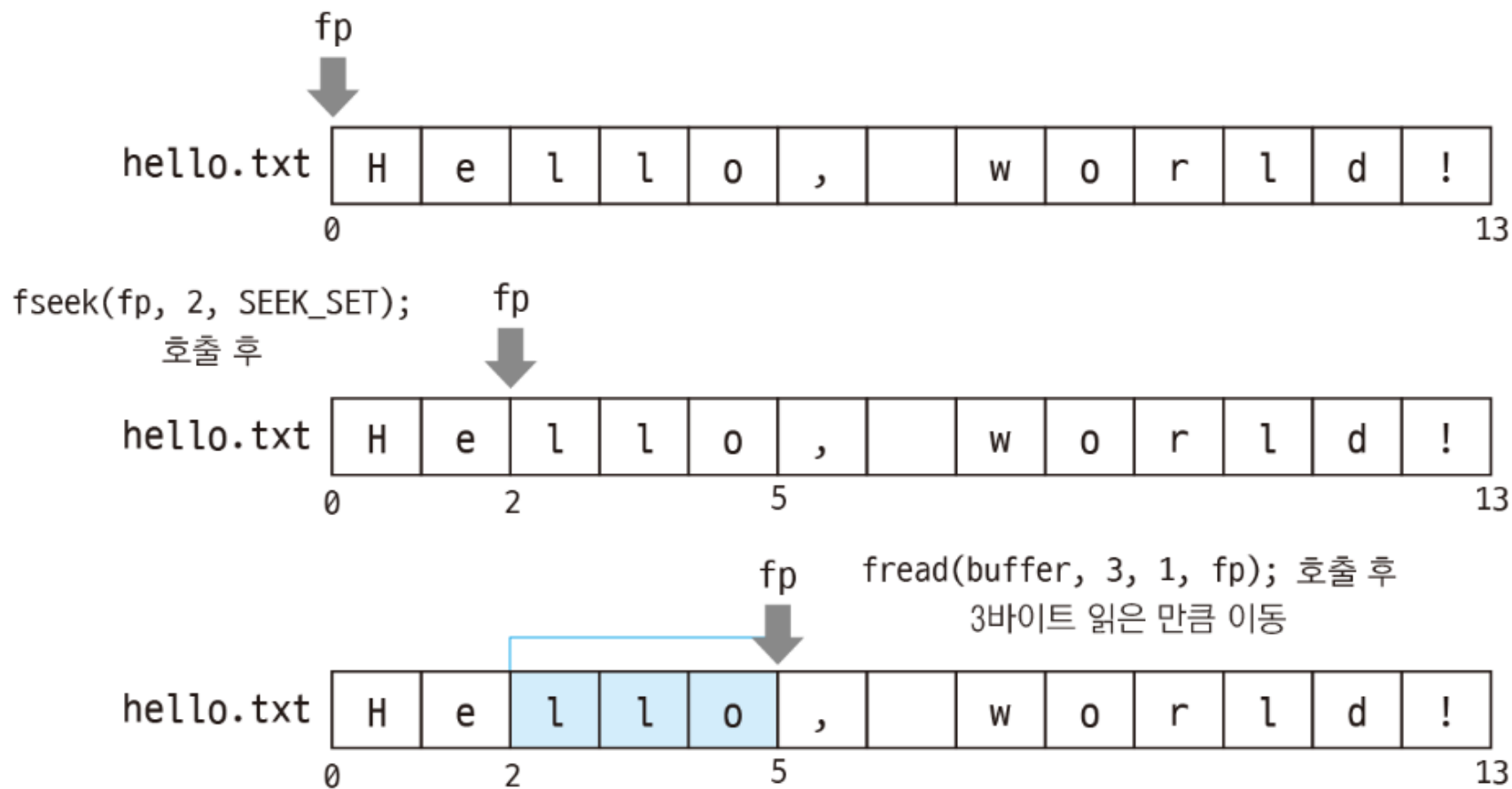
    fseek(fp, 2, SEEK_SET);            // 파일 포인터를 파일 처음에서 2바이트만큼 순방향으로 이동시킴
    fread(buffer, 3, 1, fp);           // 3바이트만큼 읽음. 3바이트만큼 순방향으로 이동
    printf("%s\n", buffer);            // llo
    memset(buffer, 0, 10);             // 버퍼를 0으로 초기화
    fseek(fp, 3, SEEK_CUR);            // 파일 포인터를 현재 위치에서 3바이트만큼 순방향으로 이동시킴
    fread(buffer, 4, 1, fp);           // 4바이트만큼 읽음. 4바이트만큼 순방향으로 이동
    printf("%s\n", buffer);            // orld
    fclose(fp);    // 파일 포인터 닫기
    return 0;
}
```

```
llo
orld
```

# 파일 포인터 활용하기

## ▶ 파일을 부분적으로 읽고 쓰기

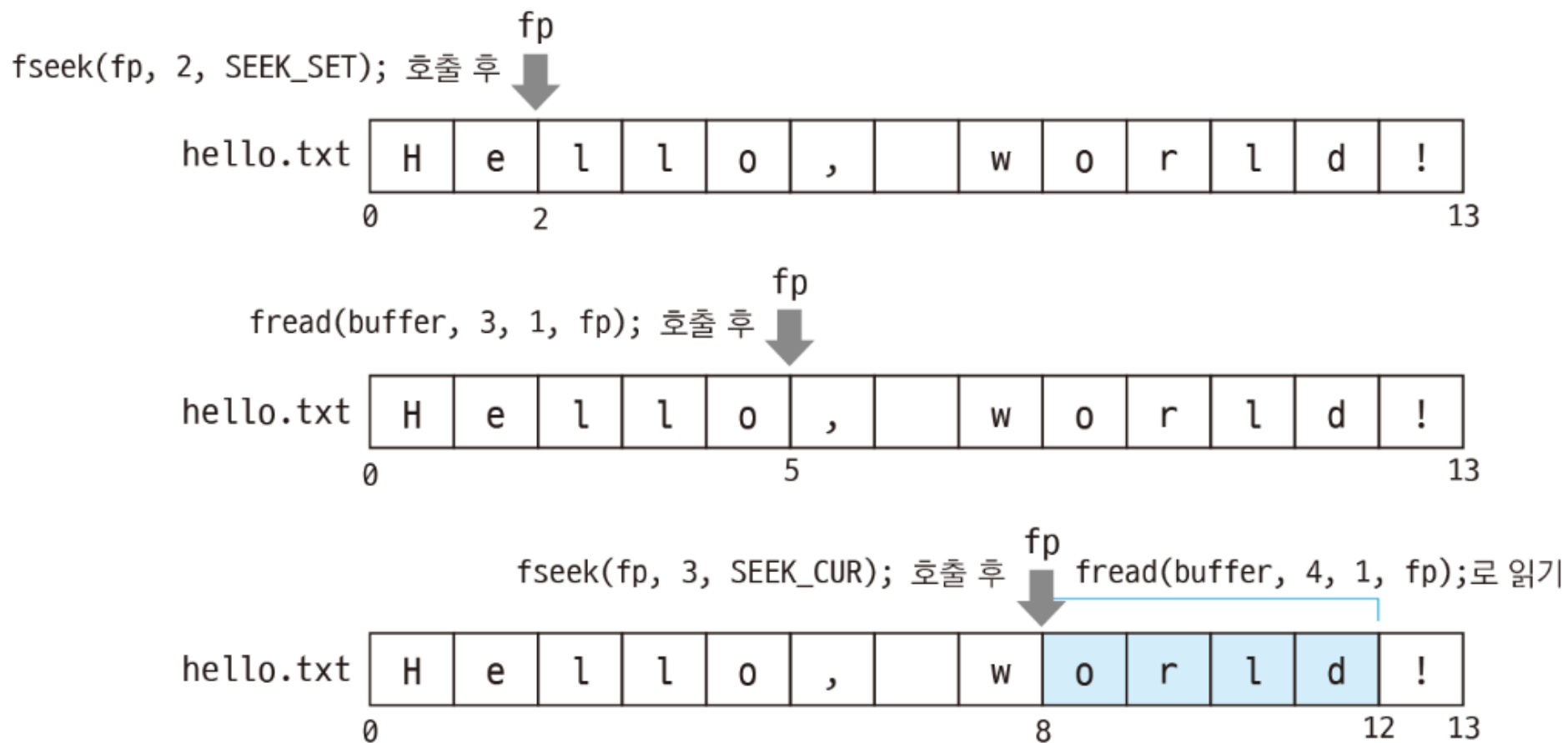
- ▶ 파일 처음에서 순방향으로 2바이트 지점의 값을 읽음



# 파일 포인터 활용하기

## ▶ 파일을 부분적으로 읽고 쓰기

### ▶ 파일 처음에서 순방향으로 8바이트 지점의 값을 읽음



# 파일 포인터 활용하기

## ▶ 파일을 부분적으로 읽고 쓰기

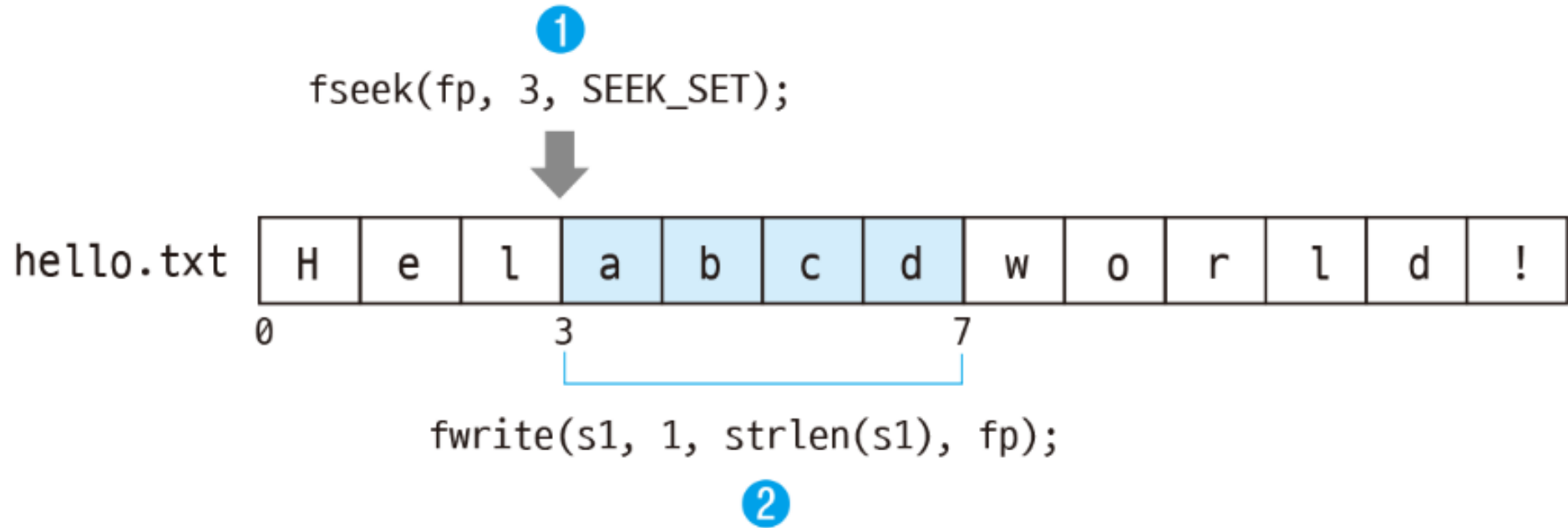
```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                // fopen, fseek, rewind, fread, fclose 함수가 선언된 헤더 파일
#include <string.h>               // strlen, memset 함수가 선언된 헤더 파일
int main(){
    char *s1 = "abcd";
    char buffer[20] = { 0, };
    FILE *fp = fopen("hello.txt", "r+");    // hello.txt 파일을 읽기/쓰기 모드(r+)로 열기.
                                           // 파일 포인터를 반환
    fseek(fp, 3, SEEK_SET);               // 파일 포인터를 파일 처음에서 3바이트만큼 순방향으로 이동시킴
    fwrite(s1, strlen(s1), 1, fp);        // 문자열 길이만큼 문자열을 파일에 저장
    rewind(fp);                           // 파일 포인터를 파일의 맨 처음으로 이동 시킴
    fread(buffer, 20, 1, fp);             // 20바이트만큼 읽음
    printf("%s\n", buffer);               // Helabcdworld!
    fclose(fp);                           // 파일 포인터 닫기
    return 0;
}
```

Helabcdworld!



# 파일 포인터 활용하기

- ▶ 파일 처음에서 순방향으로 3바이트 지점에 문자열 쓰기



# 파일 포인터 활용하기

## ▶ 제한된 버퍼로 파일 전체를 읽기

▶ feof(파일포인터);

▶ int feof(FILE \*\_Stream);

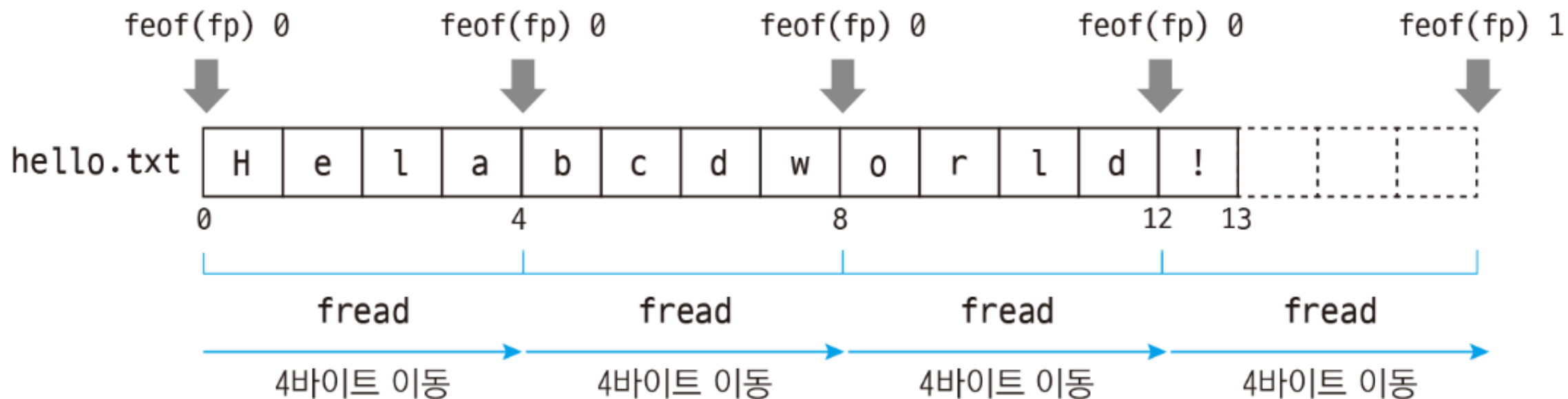
▶ feof 함수는 현재 파일 포인터가 파일의 끝인지 검사 - 파일의 끝이면 1, 끝이 아니면 0을 반환

```
#define _CRT_SECURE_NO_WARNINGS    // fopen 보안 경고로 인한 컴파일 에러 방지
#include <stdio.h>                // fopen, feof, fread, fclose 함수가 선언된 헤더 파일
#include <string.h>               // strlen, memset 함수가 선언된 헤더 파일
int main(){
    char buffer[5] = { 0, };      // 문자열 데이터 4바이트 NULL 1바이트. 4 + 1 = 5
    int count = 0;
    int total = 0;
    FILE *fp = fopen("hello.txt", "r");    // hello.txt 파일을 읽기 모드(r)로 열기.
                                           // 파일 포인터를 반환
    while (feof(fp) == 0){          // 파일 포인터가 파일의 끝이 아닐 때 계속 반복
        count = fread(buffer, sizeof(char), 4, fp);    // 1바이트씩 4번(4바이트) 읽기
        printf("%s", buffer);        // 읽은 내용 출력
        memset(buffer, 0, 5);        // 버퍼를 0으로 초기화
        total += count;              // 읽은 크기 누적
    }
    printf("\ntotal: %d\n", total);    // total: 13: 파일을 읽은 전체 크기 출력
    fclose(fp);    // 파일 포인터 닫기
    return 0;
}
```

Helabcdworld!

# 파일 포인터 활용하기

▶ 제한된 버퍼로 파일 전체를 읽기



## 퀴즈 08

▶ 파일 포인터를 파일의 맨 처음으로 이동시키려고 합니다. 올바른 코드를 고르세요. 파일 포인터는 fp입니다.

1. `fseek(fp, -10, SEEK_END);`
2. `fseek(fp, 99, SEEK_CUR);`
3. `fseek(fp, 0, SEEK_SET);`
4. `fseek(fp, 0, SEEK_END);`
5. `fseek(fp, 1, SEEK_CUR);`

## 퀴즈 09

▶ 다음 중 파일의 크기를 구하는 코드로 올바른 것을 고르세요. 버퍼는 buffer, 문자열은 s1, 파일 포인터는 fp입니다.

1. `fread(buffer, 1, -1, fp);`  
`size = ftell(fp);`
2. `fwrite(s1, 1, -1, fp);`  
`size = ftell(fp);`
3. `fseek(fp, 0, SEEK_SET);`  
`size = ftell(fp);`
4. `fseek(fp, -1, SEEK_CUR);`  
`size = ftell(fp);`
5. `fseek(fp, 0, SEEK_END);`  
`size = ftell(fp);`

# 퀴즈 10

▶ 다음 코드를 실행했을 때 파일 포인터의 위치로 올바른 것을 고르세요.

```
fseek(fp, 1, SEEK_SET);  
fread(buffer, 2, 3, fp);  
  
printf("%s\n", buffer);  
  
memset(buffer, 0, 10);  
  
fseek(fp, 8, SEEK_CUR);  
fread(buffer, 10, 1, fp);
```

1. 7
2. 15
3. 25
4. 35
5. 40

# 퀴즈 11

▶ 다음 중 파일의 끝에서 역방향으로 10바이트 지점에 문자열을 쓰는 코드로 올바른 것을 고르세요. 문자열은 `s1`, 파일 포인터는 `fp`입니다.

1. `fseek(fp, 10, SEEK_SET);`  
`fwrite(s1, strlen(s1), 1, fp);`
2. `fseek(fp, 10, SEEK_CUR);`  
`fwrite(s1, strlen(s1), 1, fp);`
3. `fseek(fp, -10, SEEK_SET);`  
`fwrite(s1, strlen(s1), 1, fp);`
4. `fseek(fp, -11, SEEK_CUR);`  
`fwrite(s1, strlen(s1), 1, fp);`
5. `fseek(fp, -10, SEEK_END);`  
`fwrite(s1, strlen(s1), 1, fp);`

# 퀴즈 12

▶ feof 함수에 대한 설명으로 올바른 것을 고르세요.

1. 파일 포인터를 파일의 맨 처음으로 이동시킴
2. 파일 포인터를 파일의 끝으로 이동시킴
3. 파일 포인터가 파일의 중간이면 1을 반환
4. 파일 포인터가 파일의 끝이면 1을 반환
5. 파일 포인터의 현재 위치를 구함



## 실습문제 03

▶ 다음 소스 코드를 완성하여 hello.txt 파일에서 읽은 문자열을 출력하세요.

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

int getFileSize(FILE *fp)
{
    int size;
    int currPos = ftell(fp);

    fseek(①_____);
    size = ftell(fp);

    fseek(fp, currPos, SEEK_SET);

    return size;
}

int main()
{
    char *buffer;
    int size;

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

    size = getFileSize(fp);
    buffer = malloc(②_____);
    memset(③_____);

    fread(buffer, size, 1, fp);

    printf("%s\n", buffer);

    fclose(fp);

    free(buffer);

    return 0;
}
```

## 실습문제 04

▶ hello.txt 파일에는 "Hello, world!"가 저장되어 있습니다. 다음 소스 코드를 완성하여 "rld, count: 3"이 출력되게 만드세요.

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

int readData(char *buffer, int offset, int size, FILE *fp)
{
    int count;

    if (feof(fp) == 1)
        return 0;

    fseek(①_____);
    count = fread(②_____);

    return count;
}

int main()
{
    char buffer[10] = { 0, };
    int count;

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

    count = readData(buffer, 9, 3, fp);

    printf("%s, count: %d\n", buffer, count);

    fclose(fp);

    return 0;
}
```

rld, count: 3

# 실습문제 05

▶ 다음 소스 코드를 완성하여 hello.txt 파일의 내용을 출력하면서 hello2.txt 파일로 복사되게 만드세요.

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

int main()
{
    int count;
    char buffer[5] = { 0, };

    FILE *src = fopen("hello.txt", "r");
    FILE *dest = fopen("hello2.txt", "w");

    while (feof(src) == 0)
    {
        ①
```

```
        printf("%s", buffer);
        ②

        memset(buffer, 0, 5);
    }

    fclose(dest);
    fclose(src);

    return 0;
}
```

## 실습문제 06

- ▶ 문자열이 저장된 words.txt 파일이 주어집니다. 다음 소스 코드에서 getData 함수를 완성하여 words.txt에서 읽은 문자열을 출력하고, 그다음 줄에 읽은 크기를 출력하는 프로그램을 완성하세요.
- ▶ 정답에는 밑줄 친 부분에 들어갈 코드만 작성해야 합니다.

▶ words.txt

```
Hello, world!
```

▶ 실행결과

```
Hello, world!  
13
```

# 실습문제 06

## ▶ 소스코드

▶ 빈칸에 들어가는 코드의 라인 수는 제한이 없음

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

int getFileSize(FILE *fp){
    int size;
    int currPos = ftell(fp);
    fseek(fp, 0, SEEK_END);
    size = ftell(fp);
    fseek(fp, currPos, SEEK_SET);
    return size;
}

char *getData(int offset, int size, int *count, FILE *fp)
{
    _____
}

int main(){
    char *buffer;
    int size;
    int count;

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

    size = getFileSize(fp);
    buffer = getData(0, size, &count, fp);

    printf("%s\n", buffer);
    printf("%d", count);

    fclose(fp);

    free(buffer);

    return 0;
}
```

# 연습문제 01

---

- ▶ 단어 두 개를 gets로 입력받아 fputs로 텍스트 파일에 저장하시오.
  - ▶ 첫 줄을 출력한 이후에는 fputs('\\n', fp)를 넣어서 줄바꿈을 하고 두번째 단어를 출력

## 연습문제 02

- ▶ first.txt와 second.txt파일을 열고 first.txt 파일의 끝에 second.txt를 이어 붙여 새로운 파일인 merge.txt에 저장하는 프로그램을 작성하시오.
- ▶ 만일 first.txt나 second.txt 파일이 없는 경우 오류 메시지를 출력하고 프로그램을 종료

## 연습문제 03

- ▶ 매크로 상수 `__FILE__` 은 C 소스코드 파일이 있는 경로를 나타낸다. 따라서 `FILE* fp = fopen(__FILE__, "r");` 이라고 하면 소스코드 자체를 열수 있다. 현재의 소스코드 파일을 읽어서 소괄호(parenthesis)의 개수와 중괄호{brace}의 개수를 출력하고 여는 괄호와 닫는 괄호 수의 수가 일치하는지 확인하시오.

```
left parenthesis : 17, right parenthesis : 17  
left brace : 6, right brace : 6
```



---

**Q & A**

---