9. File: High-level I/O

Hyunchan, Park

http://oslab.jbnu.ac.kr

Division of Computer Science and Engineering

Jeonbuk National University

학습 내용

- High-level File I/O
- Text file I/O
- Binary file I/O

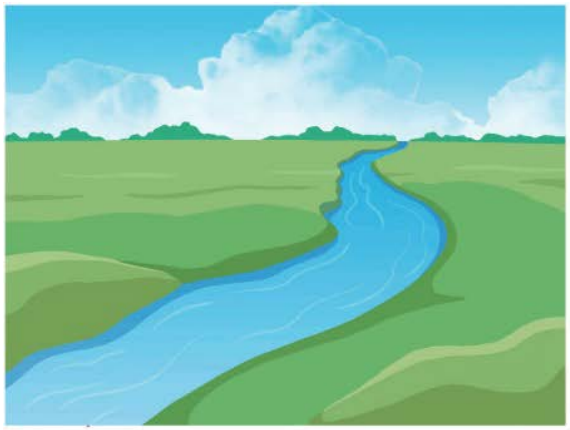
개인 과제 8: 실습

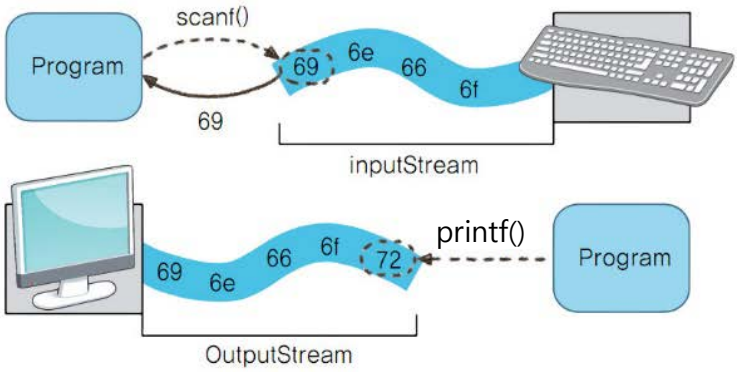
- 실습 과제
  - 실습 내용에서 등장한 프로그램들을 작성하고, 그 결과를 확인할 것
    - 결과 성공 후, cat 으로 파일의 내용을 출력할 것
    - 슬라이드에 캡처된 프로그램들은 모두 포함되어야 함
  - 제출 방법
    - Old LMS, 과제 8
    - Xshell 로그 파일 1개 제출
    - 파일 명: 학번.txt
- 제출 기한
  - 11/9 (월) 23:59 (지각 감점: 5%p / 12H, 1주 이후 제출 불가)

High-level File I/O

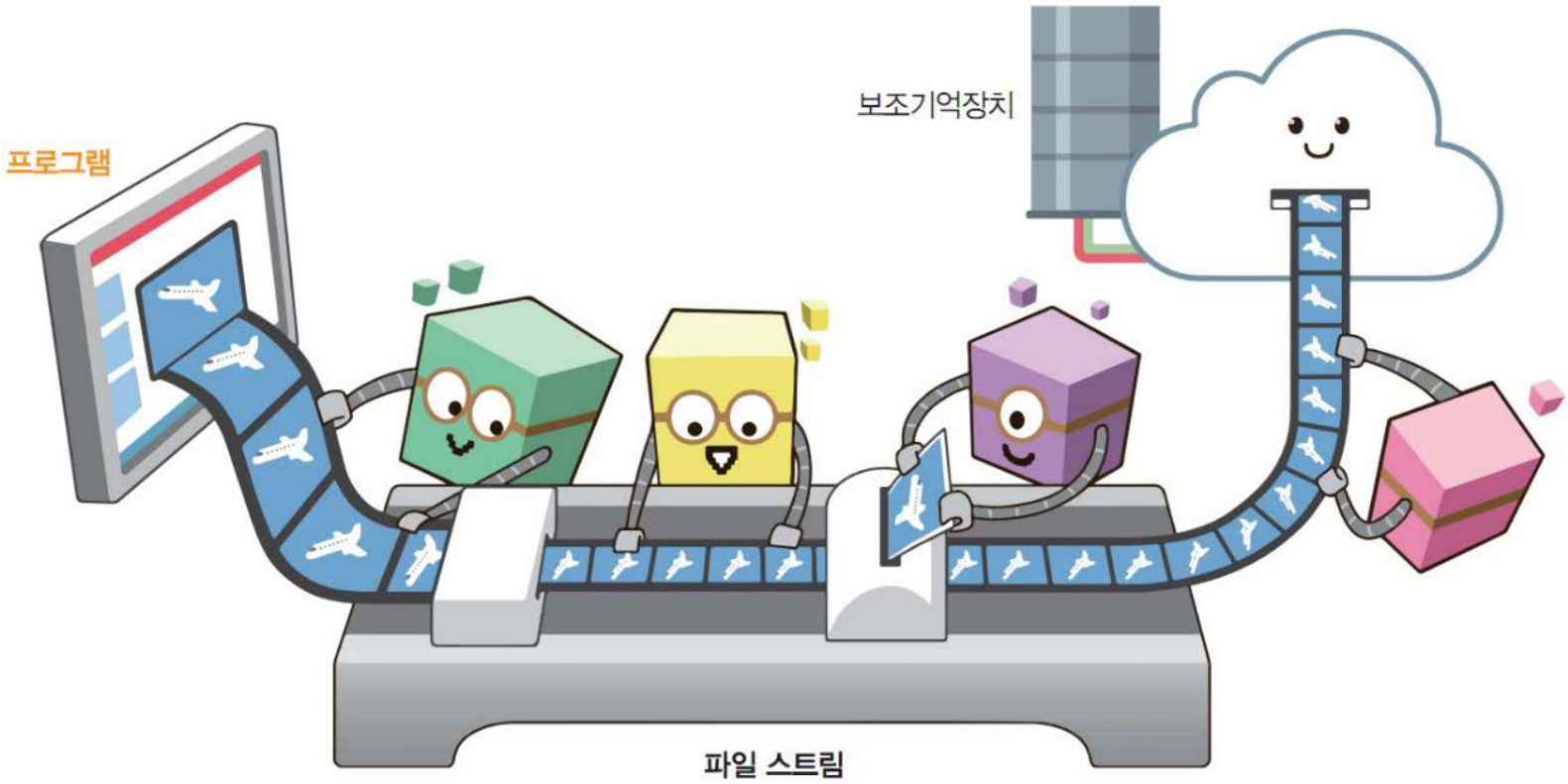
I/O Stream

- Stream: 물줄기, 개울, 시내
  - I/O stream: 데이터가 흘러가는 것. 데이터의 이동을 표현하는 말
    - 예) Media streaming
    - Source 에서 Destination 으로 데이터가 이동함
    - 프로그램은 데이터가 이동하는 통로로, source 이자 destination 일 수 있음
  - Input stream: Source로부터 데이터가 들어오는 경로
    - 예) Keyboard -> Program (scanf() 로 stream 을 연결할 수 있음)
  - Output stream: Destination으로 데이터가 나가는 경로
    - 예) Program -> Console (printf() 로 stream 을 연결할 수 있음)



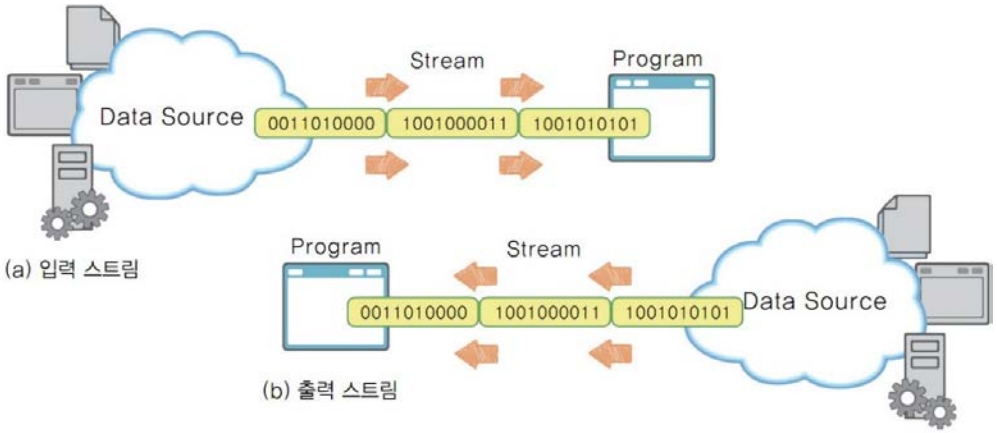


I/O Stream



I/O Stream

- 입력 스트림(input stream): 다른 곳에서 프로그램으로 들어오는 경로
  - 자료가 떠나는 시작 부분이 자료 원천부 (data source)
  - 표준입력: 원천부가 키보드
    - 파일입력: 파일이면 파일로부터 자료를 읽는 것
    - 스크린입력: 터치스크린이면 스크린에서 터치 정보
    - 네트워크입력: 다른 곳에서 프로그램으로 네트워크를 통해 자료가 전달
- 출력 스트림(output stream) : 프로그램에서 다른 곳으로 나가는 경로
  - 자료의 도착 장소가 자료 목적부 (data destination)
  - 표준출력: 목적부가 콘솔
    - 파일출력: 파일이면 파일에 원하는 값을 저장
    - 프린터출력: 프린터이면 프린터에 출력물
    - 네트워크출력: 네트워크이면 네트워크 출력이 되어 다른 곳으로 자료가 이동



텍스트 파일과 이진파일

- 텍스트 파일: 메모장(notepad) 같은 편집기로 작성된 파일
  - 내용이 아스키 코드(ascii code)와 같은 문자 코드값으로 저장
  - 메모리에 저장된 실수와 정수와 같은 내용도 문자 형식으로 변환되어 저장
  - 텍스트 편집기를 통하여 그 내용을 볼 수 있고 수정 가능
- 이진 파일: 실행파일과 그림 파일, 음악 파일, 동영상 파일 등
  - 목적에 알맞은 자료가 이진 형태(binary format)로 저장되는 파일
  - 자료는 메모리 자료 내용에서 어떤 변환도 거치지 않고 그대로 파일에 기록
  - 입출력 속도도 텍스트 파일보다 빠름
  - 메모장과 같은 텍스트 편집기로는 그 내용을 볼 수 없음
  - 내용을 이미 알고 있는 특정한 프로그램에 의해 인지될 때 의미가 있음

텍스트 파일과 이진파일

텍스트 파일

이 텍스트 파일은 메모장과 같은 텍스트 편집기를 사용해 그 내용을 볼 수 있으며 필요하면 편집도 할 수 있다.

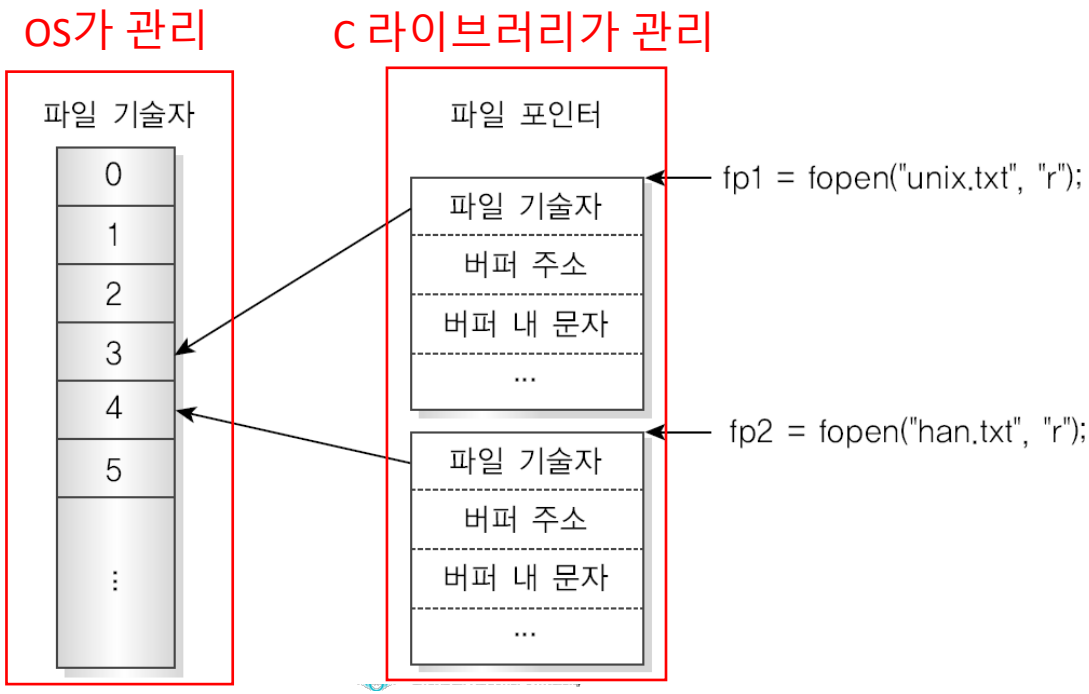
이진 파일

구조체의 변수 등 주기억장치의 내용을 그대로 저장

이미지파일, 동영상 파일, 실행파일과 같이 데이터로 구성된 파일로 일반 에디터로는 그 내용을 볼 수 없다.

파일 스트림과 포인터

- 표준 입출력 라이브러리에서 제공하는 고수준 파일 입출력
  - File stream 이라는 서비스를 사용자에게 제공한다.
    - 저수준IO: 파일 vs. 고수준IO: 파일 스트림
- FILE* : 파일 포인터
  - 고수준 파일 입출력에서 열린 파일을 가리키는 포인터
  - 자료형으로 FILE * 형을 사용 -> 구조체에 대한 포인터



파일 포인터

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

```
if ( (f = fopen(fname, "w")) == NULL )
{
   printf( "파일이 열리지 않습니다.\n" );
   exit(1);
};
```

그림 15-8 구조체 FILE과 함수 fopen()의 사용

파일 스트림 열기

- 파일 스트림 열기: fopen(3)

```
#include <stdio.h>

FILE *fopen(const char *pathname, const char *mode);
```

  - Pathname으로 지정한 파일을 mode로 지정한 모드에 따라 열고 파일 포인터를 리턴
  - mode 값

모드	의미
r	읽기 전용으로 텍스트 파일을 연다.
w	새로 쓰기용으로 텍스트 파일을 연다. 기존 내용은 삭제된다.
a	추가용으로 텍스트 파일을 연다.
rb	읽기 전용으로 바이너리 파일을 연다.
wb	새로 쓰기용으로 바이너리 파일을 연다. 기존 내용은 삭제된다.
ab	추가용으로 바이너리 파일을 연다.
r+	읽기와 쓰기용으로 텍스트 파일을 연다.
w+	쓰기와 읽기용으로 텍스트 파일을 연다.
a+	추가와 읽기용으로 텍스트 파일을 연다.
rb+	읽기와 쓰기용으로 바이너리 파일을 연다.
wb+	쓰기와 읽기용으로 바이너리 파일을 연다.
ab+	추가와 읽기용으로 바이너리 파일을 연다.

파일 스트림 열기

- 파일모드에서 +의 삽입은 수정(update) 모드 의미
  - 원래의 모드에서 읽기 또는 쓰기가 추가되는 모드
  - 수정(update) 모드에서는 모드 간의 전환이 가능
- 파일모드 r+
  - 처음에 읽기 모드로 파일을 열어 쓰기 모드로 전환 가능
  - 파일이 없으면 오류가 발생
- 파일모드 w+
  - 처음에 쓰기 모드로 파일을 열어 필요하면 읽기 모드로 전환 가능
  - 만일 파일이 존재한다면 이전의 내용은 모두 사라짐
- 파일모드 a+
  - 처음에 추가 모드로 파일을 열어 필요하면 읽기 모드로 전환 가능

파일 스트림 닫기

- 파일 스트림 닫기: fclose(3)

```
#include <stdio.h>

int fclose(FILE *stream);
```

  - Fopen() 으로 오픈한 파일 스트림을 닫는다.
  - Close() 와 마찬가지로, 파일 스트림의 사용 종료를 알리는 역할이며, 저장 장치에 내용이 기록되는 것을 보장하지는 않는다
    - 저장 장치에 즉시 기록하려면: fflush(3)를 사용
  - RETURN VALUE
    - On success: 0
    - On error: EOF
      - 성공하든, 실패하든 파일 접근은 더 이상 하면 안 됨

버퍼 기반 읽기 및 쓰기

- 버퍼 기반 입력함수: fread(3)

```
#include <stdio.h>

size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
```

  - Stream으로 지정한 파일로부터, 항목의 크기가 size인 데이터를 nitems에 지정한 개수만큼 읽어 ptr에 저장 (구조체의 입출력에 적합함)
  - 성공하면 읽어온 항목 수를 리턴
  - 읽을 항목이 없으면 0을 리턴

- 버퍼 기반 출력함수: fwrite(3)

```
#include <stdio.h>

size_t fwrite(const void *ptr, size_t size, size_t nitems, FILE *stream);
```

  - 항목의 크기가 size인 데이터를 nitems에서 지정한 개수만큼 ptr에서 읽어서 stream으로 지정한 파일에 출력
  - 성공하면 출력한 항목의 수를 리턴
  - 오류가 발생하면 EOF를 리턴

[예제 1] fread() and fwrite()

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

int main(int argc, char* argv[]) {
    FILE *rfp, *wfp;
    int i=0, count;
    char buf[80];       //80 = 1 line for standard console

    if(argc != 3) {
            printf("< Usage: ./file3 file_for_read file_for_write >\n");
            return 1;
    }

    rfp = fopen(argv[1], "r");
    if (rfp == NULL) {
        perror("Open file for read");
        exit(1);
    }

    wfp = fopen(argv[2], "w");
    if (wfp == NULL) {
        perror("Open file for write");
        exit(1);
    }

    printf("%s and %s are opened! rfp = %p wfp = %p\n", argv[1], argv[2], rfp, wfp);

    while ((count = fread(buf, 1, 10, rfp)) > 0 ) {
            printf("%d: count=%d\n", i++, count);
            fwrite(buf, 1, count, wfp);
    }

    fclose(rfp);
    fclose(wfp);

    return 0;
}
```

생성한 파일의 접근 권한은?

1B 단위로, 10개 항목씩 읽어옴

파일 오프셋 지정[1]

- 파일 오프셋 이동: fseek(3)

```
#include <stdio.h>

int fseek(FILE *stream, long offset, int whence);
```

  - stream이 가리키는 파일에서 offset에 지정한 크기만큼 오프셋을 이동
  - whence는 lseek()와 같은 값을 사용
  - 성공하면 0을, 실패하면 EOF를 리턴

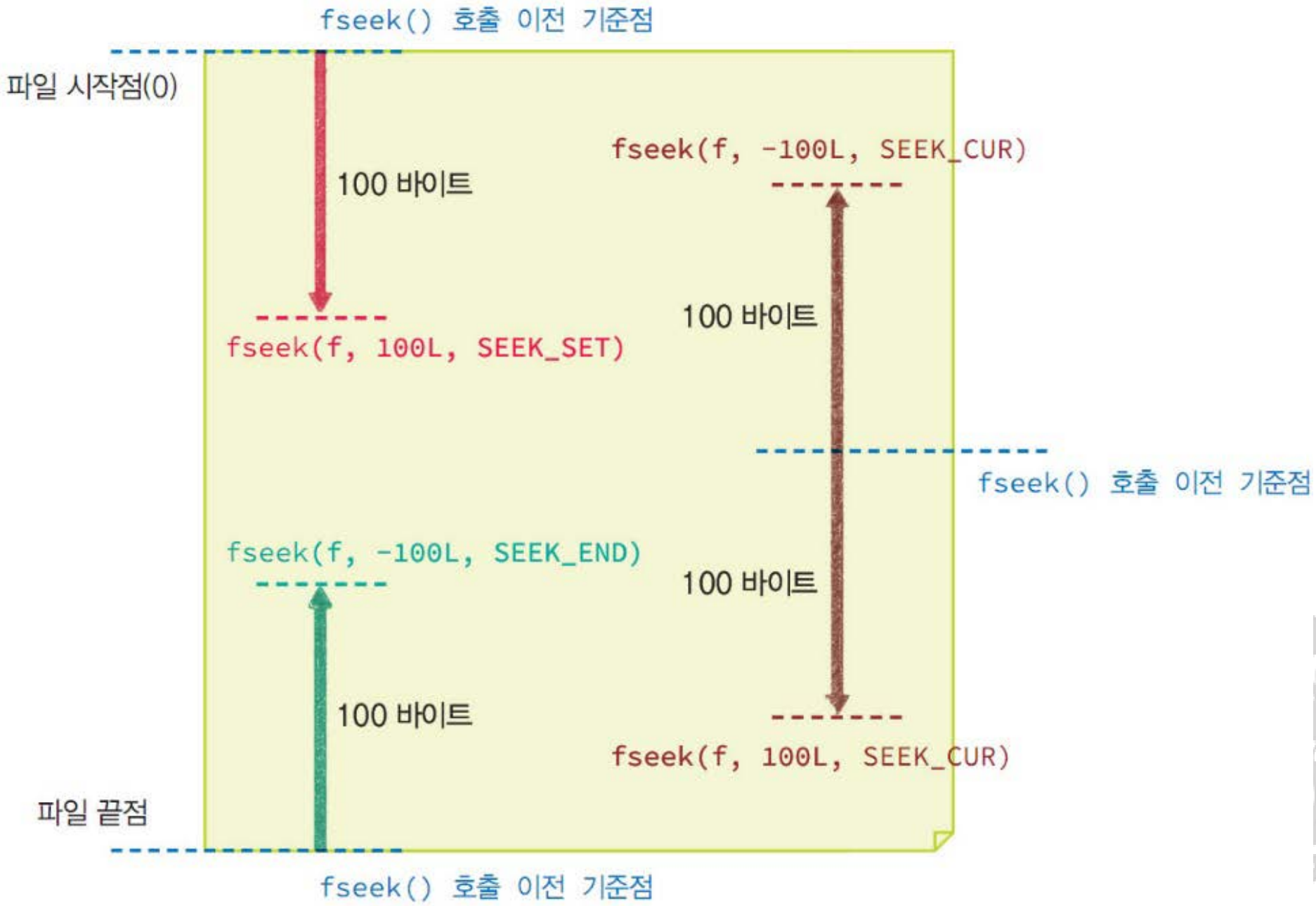
- 현재 오프셋 구하기: ftell(3)

```
#include <stdio.h>

long ftell(FILE *stream);
```

  - 현재 오프셋을 리턴. 오프셋은 파일의 시작에서 현재 위치까지의 바이트 수

파일 오프셋 지정[2]

- 함수 fseek(f, 100L, SEEK_SET)의 호출
  - 파일 위치를 파일의 처음 위치에서 100바이트 떨어진 위치로 이동
- 함수 fseek(f, 100L, SEEK_CUR)의 호출
  - 파일의 현재 위치에서 100바이트 떨어진 위치로 이동
- 함수 fseek(f, -100L, SEEK_END)의 호출
  - 파일 끝 위치에서 앞으로 100바이트 떨어진 위치로 이동
- 함수 fseek()에서 offset
  - 양수이면 파일의 끝점으로,
  - 음수이면 파일의 시작점으로의 이동방향을 표시


fseek() 호출 이전 기준점
파일 시작점(0)
100 바이트
fseek(f, 100L, SEEK_SET)
fseek(f, -100L, SEEK_CUR)
100 바이트
fseek() 호출 이전 기준점
fseek(f, -100L, SEEK_END)
100 바이트
100 바이트
fseek(f, 100L, SEEK_CUR)
파일 끝점
fseek() 호출 이전 기준점

파일 오프셋 지정[3]

- 처음 위치로 오프셋 이동: rewind(3)

```
#include <stdio.h>

void rewind(FILE *stream);
```

  - 오프셋을 파일의 시작 위치로 즉시 이동

- 오프셋의 저장과 이동: fsetpos(3), fgetpos(3)

```
#include <stdio.h>

int fsetpos(FILE *stream, const fpos_t *pos);
int fgetpos(FILE *stream, fpos_t *pos);
```

  - fgetpos() : 파일의 현재 오프셋을 pos가 가리키는 영역에 저장
  - fsetpos() : pos가 가리키는 위치로 파일 오프셋을 이동

- (기억해둘 필요가 있을까?)

[예제 2] fseek() and ftell()

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

char buf[80];
int count;
FILE* fp;

void read_five_bytes(void) {
    if ((count = fread(buf, 1, 5, fp)) <= 0 ) {
        perror("Read Error");
        exit(1);
    }
}
```

```
ubuntu@41983:~/hw3$ ./high2 unix.txt
unix.txt is opened! fp = 0x55797d3a12a0

5: hello
Current position: 5

5: ello
Current position: 6

5: llo w
Current position: 7
```

```c
int main(int argc, char* argv[]) {
    if(argc != 2) {
            printf("< Usage: ./high2 filename >\n");
            return 1;
    }

    fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror("Open");
        exit(1);
    }

    printf("%s is opened! fp = %p\n", argv[1], fp);

    read_five_bytes();
    printf("\n%d: %s\n", count, buf);
    printf("Current position: %ld\n", ftell(fp));

    fseek(fp,1,SEEK_SET);
    read_five_bytes();
    printf("\n%d: %s\n", count, buf);
    printf("Current position: %ld\n", ftell(fp));

    fseek(fp,2,SEEK_SET);
    read_five_bytes();
    printf("\n%d: %s\n", count, buf);
    printf("Current position: %ld\n", ftell(fp));

    fclose(fp);

    return 0;
}
```

Text file I/O

fprintf() and fscanf()

- 함수 fprintf()와 fscanf() 또는 fscanf_s()를 이용
  - 텍스트 파일에 자료를 쓰거나 읽기 위하여
  - 헤더 파일 stdio.h를 포함
    - 첫 번째 인자는 입출력에 이용될 파일
    - 두 번째 인자는 입출력에 이용되는 제어 문자열
    - 다음 인자들은 입출력될 변수 또는 상수 목록

```
#include <stdio.h>

extern FILE *stdin;
extern FILE *stdout;
extern FILE *stderr;
```

```
#include <stdio.h>

int fprintf(FILE *stream, const char *format, ...);

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

  - 함수 fprintf()와 fscanf() 또는 fscanf_s()의 첫 번째 인자에 각각 stdin 또는 stdout를 이용하면 표준 입력, 표준 출력으로 이용이 가능
  - Return value
    - fprintf(): 기록한 문자 수. 문자열 마지막의 NULL 문자는 제외
    - Fscanf(): 읽은 아이템 수 혹은 파일의 끝에 도달한 경우 EOF

[예제 3] fprintf()

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

int main(int argc, char* argv[]) {
    FILE *fp;

    if(argc != 2) {
            printf("< Usage: ./high3 filename >\n");
            return 1;
    }

    fp = fopen(argv[1], "w");
    if (fp == NULL) {
        perror("Open");
        exit(1);
    }

    fprintf(fp, "%s %d %d %d %2.2f\n","2015123", 80, 90, 100, 90.00);
    fprintf(fp, "%s %d %d %d %2.2f\n","2016123", 50, 80, 60, 66.22);
    fprintf(fp, "%s %d %d %d %2.2f\n","2017123", 70, 20, 70, 55.55);

    fclose(fp);

    return 0;
}
```

```
ubuntu@41983:~/hw3$ vi high3.c
ubuntu@41983:~/hw3$ gcc -o high3 high3.c
ubuntu@41983:~/hw3$ ./high3 grade.txt
ubuntu@41983:~/hw3$ cat grade.txt
2015123 80 90 100 90.00
2016123 50 80 60 66.22
2017123 70 20 70 55.55
```

[예제 3] fprintf()

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

int main(int argc, char* argv[]) {

    fprintf(stdout, "%s %d %d %d %2.2f\n","2015123", 80, 90, 100, 90.00);
    fprintf(stdout, "%s %d %d %d %2.2f\n","2016123", 50, 80, 60, 66.22);
    fprintf(stdout, "%s %d %d %d %2.2f\n","2017123", 70, 20, 70, 55.55);

    return 0;
}
```

```
ubuntu@41983:~/hw3$ vi high3.c
ubuntu@41983:~/hw3$ gcc -o high3 high3.c
ubuntu@41983:~/hw3$ ./high3
2015123 80 90 100 90.00
2016123 50 80 60 66.22
2017123 70 20 70 55.55
ubuntu@41983:~/hw3$ ./high3 > grade.txt
ubuntu@41983:~/hw3$ cat grade.txt
2015123 80 90 100 90.00
2016123 50 80 60 66.22
2017123 70 20 70 55.55
ubuntu@41983:~/hw3$
```

[예제 4] fscanf()

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

int main(int argc, char* argv[]) {
    FILE *fp;
    char id[10];
    int grade1, grade2, grade3;
    float avg;

    if(argc != 2) {
            printf("< Usage: ./high4 filename >\n");
            return 1;
    }

    fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror("Open");
        exit(1);
    }

    fscanf(fp, "%s %d %d %d %f\n", id, &grade1, &grade2, &grade3, &avg);
    fprintf(stdout, "%s %d %d %d %.3f\n", id, grade1, grade2, grade3, avg);

    fscanf(fp, "%s %d %d %d %f\n", id, &grade1, &grade2, &grade3, &avg);
    fprintf(stdout, "%s %d %d %d %.3f\n", id, grade1, grade2, grade3, avg);

    fscanf(fp, "%s %d %d %d %f\n", id, &grade1, &grade2, &grade3, &avg);
    fprintf(stdout, "%s %d %d %d %.3f\n", id, grade1, grade2, grade3, avg);

    fclose(fp);


    return 0;
}
```

```
ubuntu@41983:~/hw3$ gcc -o high4 high4.c
ubuntu@41983:~/hw3$ ./high4 grade.txt
2015123 80 90 100 90.000
2016123 50 80 60 66.220
2017123 70 20 70 55.550
```

feof() and ferror()

- 함수 feof(): 파일 스트림의 EOF(End Of File) 표시를 검사하는 함수
  - 읽기 작업이 파일의 이전 부분을 읽으면 0을 반환하고 (EOF 아님)
    - 그렇지 않으면 0이 아닌 값을 반환 (파일 끝!)
    - 이전 읽기 작업에서 EOF 표시에 도달하면 0이 아닌 값으로 지정 (파일 끝!)
  - 단순히 파일 지시자가 파일의 끝에 있더라도 feof()의 결과는 0
    - 파일 끝에 도달한 다음, 한 번 더 읽기 동작이 수행되어야 EOF 가 셋팅됨
    - 이 동작 때문에 실제 사용 시, 잘못 사용하는 경우가 많음 ~~(쓰지마!)~~
- 함수 ferror(): 파일 처리에서 오류가 발생했는지 검사하는 함수
  - 이전 파일 처리에서 오류가 발생하면 0이 아닌 값을 반환 (오류!)
    - 오류가 발생하지 않으면 0을 반환 (정상)
  - 헤더파일 stdio.h 필요

```
#include <stdio.h>

void clearerr(FILE *stream);

int feof(FILE *stream);

int ferror(FILE *stream);
```

[예제 5] fscanf() with feof()

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

int main(int argc, char* argv[]) {
    FILE *fp;
    char id[10];
    int grade1, grade2, grade3;
    float avg;

    if(argc != 2) {
            printf("< Usage: ./high5 filename >\n");
            return 1;
    }

    fp = fopen(argv[1], "r");
    if (fp == NULL) {
        perror("Open");
        exit(1);
    }

    while(!feof(fp)) {
        fscanf(fp, "%s %d %d %d %f\n", id, &grade1, &grade2, &grade3, &avg);
        fprintf(stdout, "%s %d %d %d %.3f\n", id, grade1, grade2, grade3, avg);
    }

    fclose(fp);


    return 0;
}
```

```
ubuntu@41983:~/hw3$ gcc -o high5 high5.c
ubuntu@41983:~/hw3$ ./high5 grade.txt
2015123 80 90 100 90.000
2016123 50 80 60 66.220
2017123 70 20 70 55.550
```

Additional services: fgetc() and fputc()

```
#include <stdio.h>

int fgetc(FILE *stream);

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

- 함수 fgetc()
  - 파일로부터 문자 하나를 입력받는 함수
  - Return value
    - int 형 으로 casting 된 문자 하나를 반환
    - 만약 파일 끝인 경우, EOF 를 반환하며, 에러에는 0이 아닌 값
- 함수 fputc()
  - 문자 하나를 파일로 출력하는 함수
  - 함수들은 문자 하나의 입출력의 대상인 파일 포인터를 인자로 이용
- Getc()와 putc()도 존재함. 그러나 잊어버릴 것.
  - Macro 로 구현되어 있을 가능성이 있어, fgetc(), fputc() 보다 빠를 수도 있지만, 예상치 못한 문제점이 생길 수 있음

Additional services: fgets() and fputs()

```
#include <stdio.h>

char *fgets(char *s, int size, FILE *stream);

int fputs(const char *s, FILE *stream);
```

- 함수 fgets(): 파일로부터 한 행의 문자열을 입력 받는 함수
  - 파일로부터 문자열을 개행문자(\n)까지 읽어 마지막 개행문자를 '\0'문자로 바꾸어 입력 버퍼 문자열에 저장
    - 첫 번째 인자는 문자열이 저장될 문자 포인터
    - 두 번째 인자는 입력할 문자의 최대 수
    - 세 번째 인자는 입력 문자열이 저장된 파일
  - 텍스트 파일을 처리할 때, 라인 별로 처리가 가능하여 아주 편리함
- 함수 fputs(): 파일로 한 행의 문자열을 출력하는 함수
  - 문자열을 한 행에 출력
    - 첫 번째 인자는 출력될 문자열이 저장된 문자 포인터
    - 두 번째 인자는 문자열이 출력되는 파일
  - fprintf() 면 충분하지 않을까?

[예제 6-1] fgets() with feof() FAILED!!

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

#define MAX_BUF 80

int main(int argc, char* argv[]) {
    FILE *fp;
    int line=0;
    char buf[MAX_BUF];

    if(argc != 2) {
            printf("< Usage: ./high6 filename >\n");
            return 1;
    }

    if ((fp = fopen(argv[1], "r")) == NULL) {
        perror("Open");
        exit(1);
    }

    while (!feof(fp)) {
            fgets(buf, MAX_BUF, fp);
            fprintf(stdout, "%3d: %s", line++, buf);
    }

    fclose(fp);

    return 0;
}
```

```
ubuntu@41983:~/hw3$ cat unix.txt
hello world
ubuntu@41983:~/hw3$ gcc -o high6-1 high6-1.c
ubuntu@41983:~/hw3$ ./high6-1 unix.txt
  0: hello world
  1: hello world
ubuntu@41983:~/hw3$
```

이렇게 logic을 작성하는 경우가 많으나 의도와 다르게 동작하는 코드.
예제에서 Loop 내의 코드가 몇 번 실행될까?

[예제 6-2] fgets()

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

#define MAX_BUF 80

int main(int argc, char* argv[]) {
    FILE *fp;
    int line=0;
    char buf[MAX_BUF];

    if(argc != 2) {
            printf("< Usage: ./high6 filename >\n");
            return 1;
    }

    if ((fp = fopen(argv[1], "r")) == NULL) {
        perror("Open");
        exit(1);
    }

    while (fgets(buf, MAX_BUF, fp) != NULL) {
            fprintf(stdout, "%3d: %s", line++, buf);
    }

    fclose(fp);

    return 0;
}
```

```
ubuntu@41983:~/hw3$ vi high6.c
ubuntu@41983:~/hw3$ gcc -o high6 high6.c
ubuntu@41983:~/hw3$ ./high6 unix.txt
  0: hello world
```

[예제 6] fgets()

```
ubuntu@41983:~/hw3$ ./high6 novel.txt
  0:    Once upon a time . . . there were three little pigs, who left their mummy
  1: and daddy to see the world.
  2:    All summer long, they roamed through the woods and over the plains,playing
  3: games and having fun. None were happier than the three little pigs, and they
  4: easily made friends with everyone. Wherever they went, they were given a warm
  5: welcome, but as summer drew to a close, they realized that folk were drifting
  6: back to their usual jobs, and preparing for winter. Autumn came and it began
  7: to rain. The three little pigs started to feel they needed a real home. Sadly
  8: they knew that the fun was over now and they must set to work like the others.
```

```
 70:    The flames licked his hairy coat and his tail became a flaring torch.
 71:    "Never again! Never again will I go down a chimneyl" he squealed, as he
 72: tried to put out the flames in his tail. Then he ran away as fast as he could.
 73:    The three happy little pigs, dancing round and round the yard, began to
 74: sing:
 75:    "Tra-la-la! Tra-la-la! The wicked black wolf will never come back...!"
 76:    From that terrible day on, the wisest little pig's brothers set to work
 77: with a will. In less than no time, up went the two new brick houses. The wolf
 78: did return once to roam in the neighbourhood, but when he caught sight of
 79: three chimneys, he remembered the terrible pain of a burnt tail, and he left
 80: for good.
 81:    Now safe and happy, the wisest little pig called to his brothers:
 82:    "No more work! Come on, let's go and play!"
ubuntu@41983:~/hw3$
```

Binary file I/O

Text vs. Binary

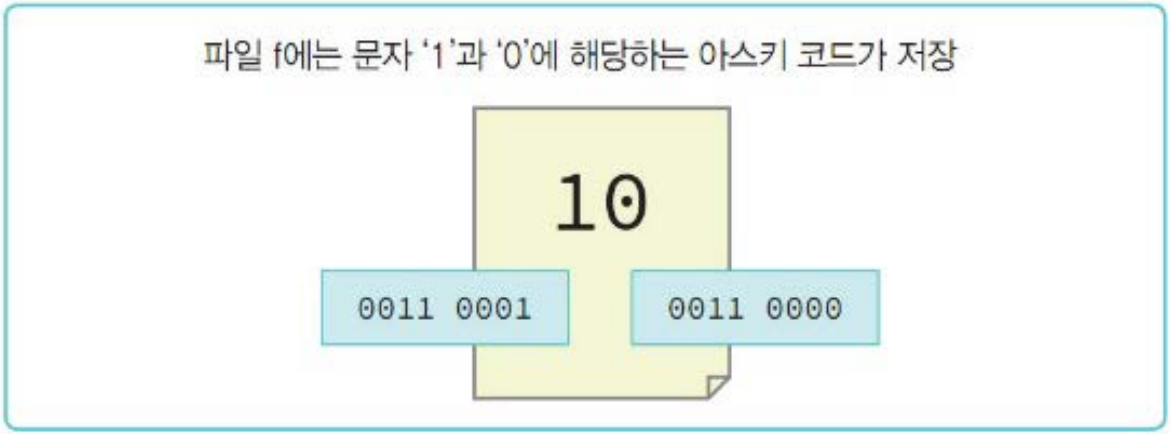
- Text I/O: fscanf() and fprintf()
  - 본래 파일은 “Collection of Bytes”
  - 텍스트 파일은? 자료의 입출력을 텍스트 모드(아스키 코드)로 처리
    - 텍스트 파일의 내용은 모두 지정된 아스키 코드와 같은 문자 코드값
  - 함수 fprintf()를 이용
    - int 형 변수 cnt의 값을 파일 f에 출력하는 과정
    - 실제로 파일에 저장되는 자료는 정수값 10에 해당하는 각 문자의 아스키 값
      - 각각의 문자 ‘1’과 ‘0’을 아스키 코드값으로 변환: 0x31, 0x30
      - 변환한 결과값을 저장함

```
int cnt = 10;

fprintf(f, "%d", cnt);
```

```
int cnt = 10;

fprintf(f, "%d", cnt);
```

메모리

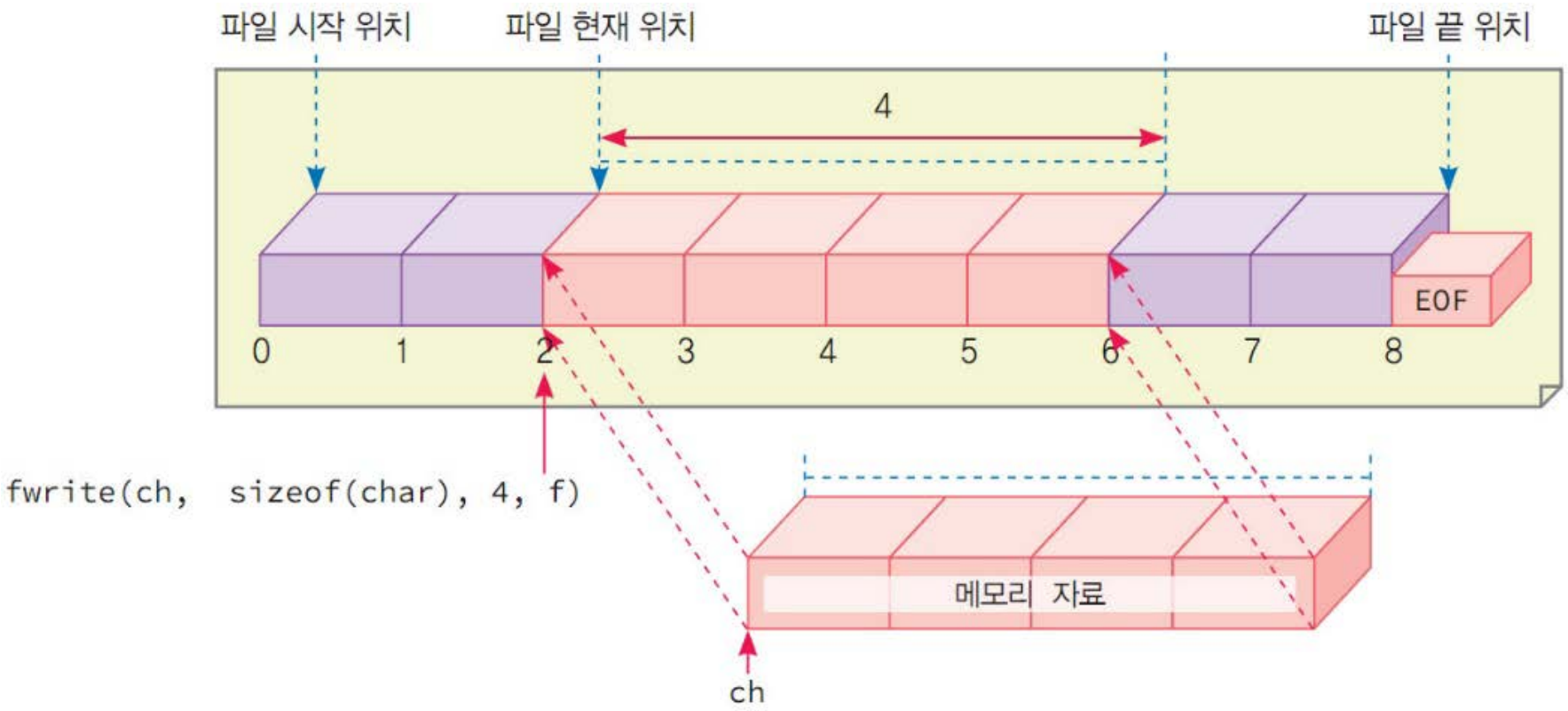
파일



Text vs. Binary

- Binary I/O: fread() and fwrite()
  - C 언어의 자료형을 유지하면서, 변환없이 그대로 바이트 단위로 저장
  - 입출력은 Low-level I/O(read() and write())와 같이 바이트 단위로 수행되지만,
  - 자료형에 따라 데이터 개체 단위로 이용할 수 있도록 인터페이스가 다름
    - Low-level I/O: file, buf, size
    - High-level I/O: file, buf, size and number of data objects
  - 예) char[4] 형 자료의 쓰기
    - char name[4]; // sizeof(name) = 4
    - Low-level: write(fd, name, 4); // 4B 사이즈의 데이터
    - High-level: fwrite(name, 4, 1, fp); // 4B 사이즈의 데이터를 1개 쓰기
  - 구조체 데이터를 읽고 쓰기에 적합함
    - C에서는 연관있는 데이터를 구조체로 엮어서 사용하는 경우가 많고,
    - Binary I/O 는 Text I/O 보다 성능 및 용량 면에서 효율적으로 I/O가 가능함

Text vs. Binary



[예제 7] 구조체 파일 쓰기

- 구조체 student: 학생의 성적 정보를 구조체로 표현
  - 학번, 과목1, 과목2, 과목3, 평균 점수를 멤버로 구성

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

typedef struct {
    char id[10];
    int grade1, grade2, grade3;
    float avg;
} student;
```

- 표준입력으로 3명의 학생 자료를 입력 받은 구조체 3개를 파일 “unix.bin"에 저장하는 프로그램
  - 표준입력은 학생마다 한 행 씩 입력 받고,
  - 함수 fscanf() 를 이용하여 문자열에서 자료를 추출해 구조체에 저장하고,
  - 구조체 내용을 이진 파일로 기록함

[예제 7] 구조체 파일 쓰기

```c
int main(int argc, char* argv[]) {
    FILE *fp;
    int i;
    student data[3];

    if(argc != 2) {
            printf("< Usage: ./high7 filename >\n");
            return 1;
    }

    if ((fp = fopen(argv[1], "w")) == NULL) {
        perror("Open");
        exit(1);
    }

    for(i=0; i<3; i++) {
        if(fscanf(stdin, "%s %d %d %d", data[i].id, &data[i].grade1, &data[i].grade2, &data[i].grade3) > 0) {
            data[i].avg = (data[i].grade1+data[i].grade2+data[i].grade3)/3;
            fwrite(&data[i], sizeof(student), 1, fp);
        } else {
            break;
        }
    }

    fclose(fp);

    return 0;
}
```

← 구조체 개체를 fp로 하나씩 저장

[예제 7] 구조체 파일 쓰기

```
ubuntu@41983:~/hw3$ vi grade.txt
ubuntu@41983:~/hw3$ cat grade.txt
2015123 80 90 100
2016123 50 80 60
2017123 70 20 70
ubuntu@41983:~/hw3$ gcc -o high7 high7.c
ubuntu@41983:~/hw3$ cat grade.txt | ./high7 unix.bin
ubuntu@41983:~/hw3$ cat unix.bin
2015123PZd´B20161232P<|B2017123FFTBubuntu@41983:~/hw3$
ubuntu@41983:~/hw3$ vi unix.bin
```

(vi 실행 후, :%!xxd 입력하여 hex 편집 모드로 전환)

```
00000000: 3230 3135 3132 3300 0000 0000 5000 0000  2015123.....P...
00000010: 5a00 0000 6400 0000 0000 b442 3230 3136  Z...d......B2016
00000020: 3132 3300 0000 0000 3200 0000 5000 0000  123.....2...P...
00000030: 3c00 0000 0000 7c42 3230 3137 3132 3300  <.....|B2017123.
00000040: 0000 0000 4600 0000 1400 0000 4600 0000  ....F.......F...
00000050: 0000 5442 0a                             ..TB.
```

[예제 7] 구조체 파일 쓰기

```
3230 3135 3132 3300 0000 0000 5000 0000  2015123.....P...
5a00 0000 6400 0000 0000 b442 3230 3136  Z...d......B2016
3132 3300 0000 0000 3200 0000 5000 0000  123.....2...P...
3c00 0000 0000 7c42 3230 3137 3132 3300  <.....|B2017123.
0000 0000 4600 0000 1400 0000 4600 0000  ....F.......F...
0000 5442 0a                             ..TB.
```

- 10, 4, 4, 4, 4 = 26
- 그러나 현재 구조체 크기는?

  `sizeof(student) = 28`

- Memory alignment!!
  - 인텔 CPU는 데이터의 메모리 시작 주소를 항상 4의 배수가 되도록 정렬함
  - 이로 인해 10B 크기의 char id[10] 데이터가 저장된 다음, 2B 만큼의 padding space 가 생긴 것
  - 다음 int grade1 은 12B 위치부터 저장됨. 이후에는 모든 데이터가 memory aligned 된 위치이므로 padding 이 없음

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

typedef struct {
    char id[10];
    int grade1, grade2, grade3;
    float avg;
} student;
```

[예제 8] 이진 파일 읽기

```
ubuntu@41983:~/hw3$ gcc -o high8 high8.c
ubuntu@41983:~/hw3$ ./high8 unix.bin
2015123 80 90 100 90.00
2016123 50 80 60 63.00
2017123 70 20 70 53.00
ubuntu@41983:~/hw3$
```

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

typedef struct {
    char id[10];
    int grade1, grade2, grade3;
    float avg;
} student;

int main(int argc, char* argv[]) {
    FILE *fp;
    int i;
    student data[3];

    if(argc != 2) {
            printf("< Usage: ./high8 filename >\n");
            return 1;
    }

    if ((fp = fopen(argv[1], "r")) == NULL) {
        perror("Open");
        exit(1);
    }

    while(1) {
        if(fread(&data[i], sizeof(student), 1, fp) == 1) {
            fprintf(stdout, "%s %d %d %d %.2f\n", data[i].id, data[i].grade1, data[i].grade2, data[i].grade3, data[i].avg);
        } else {
            break;
        }
    }

    fclose(fp);

    return 0;
}
```