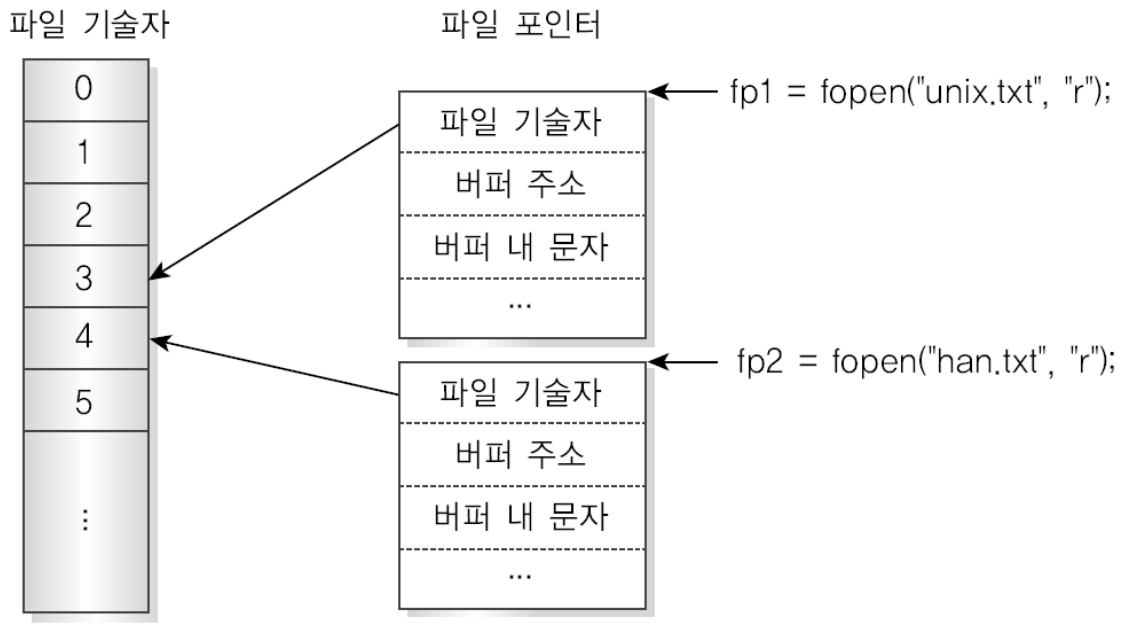
고수준 파일 입출력

파일 포인터

고수준 파일 입출력 : 표준 입출력 라이브러리

파일 포인터

- 고수준 파일 입출력에서 열린 파일을 가리키는 포인터
- 자료형으로 FILE * 형을 사용 -> 구조체에 대한 포인터



[그림 2-2] 파일 기술자와 파일 포인터

파일 열기와 닫기[1]

파일 열기: fopen(3)

```
#include <stdio.h>
FILE *fopen(const char *filename, const char *mode);
```

- filename으로 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.

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

```
r+ : 파일이 없는 경우 에러
w+ : 파일이 없는 경우 생성
```

파일 열기와 닫기[2]

파일 닫기: fclose(3)

```
#include <stdio.h>
int fclose(FILE *stream);
```

- fopen으로 오픈한 파일을 닫음

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

문자 기반 입출력 함수

문자 기반 입력함수: fgetc(3), getc(3), getchar(3), getw(3)

```
#include <stdio.h>
int fgetc(FILE *stream);
int getc(FILE *stream);
int getchar(void);
int getw(FILE *stream);
```

- fgetc : 문자 한 개를 unsigned char 형태로 읽어 옴
- getc, getchar : 매크로
- getw : 워드 단위로 읽어 옴

문자 기반 출력함수: fputc(3), putc(3), putchar(3), putw(3)

```
#include <stdio.h>
int fputc(int c, *stream);
int putc(int c, *stream);
int putchar(int c);
int putw(int w, FILE *stream);
```

문자열 기반 입출력

문자열 기반 입력 함수: gets(3), fgets(3)

```
#include <stdio.h>
char *gets(char *s);
char *fgets(char *s, int n, FILE *stream);
```

- gets : 표준 입력에서 문자열을 읽음
- fgets : 파일(stream)에서 n보다 하나 적게 문자열을 읽어 s에 저장

문자열 기반 출력 함수: puts(3), fputs(3)

```
#include <stdio.h>
char *puts(const char *s);
char *fputs(const char *s, FILE *stream);
```

버퍼 기반 입출력

버퍼 기반 입력함수: fread(3)

```
#include <stdio.h>
size_t fread(void *ptr, size_t size, size_t nitems, FILE *stream);
```

- 항목의 크기가 size인 데이터를 nitems에 지정한 개수만큼 읽어 ptr에 저장
- 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

fread/fwrite 함수 예

```
01  #include <stdlib.h>
02  #include <stdio.h>
03
04  int main(void) {
05      FILE *rfp, *wfp;
06      char buf[BUFSIZ];
07      int n;
08
09      if ((rfp = fopen("unix.txt", "r")) == NULL) {
10          perror("fopen: unix.txt");
11          exit(1);
12      }
13
14      if ((wfp = fopen("unix.out", "a")) == NULL) {
15          perror("fopen: unix.out");
16          exit(1);
17      }
18
19      while ((n = fread(buf, sizeof(char)*2, 3, rfp)) > 0) {
20          fwrite(buf, sizeof(char)*2, n, wfp);
21      }
22
23      fclose(rfp);
24      fclose(wfp);
25
26      return 0;
27  }
```

항목크기가 char크기의 2배, 이것을 3개,
즉 2*3=6바이트씩 읽어서 출력

```
# ex2_14.out
# cat unix.out
Unix System Programming
Unix System Programming
Unix System Programming
```

형식 기반 입출력

형식 기반 입력 함수: scanf(3), fscanf(3)

```
#include <stdio.h>
int scanf(const char *restrict format, ...);
int fscanf(FILE *restrict stream, const char *restrict format, ..);
```

- fscanf도 scanf가 사용하는 형식 지정 방법을 그대로 사용
- 성공하면 읽어온 항목의 개수를 리턴

형식 기반 출력 함수: printf(3), fprintf(3)

```
#include <stdio.h>
int printf(const char *restrict format, /* args */ ...);
int fprintf(FILE *restrict stream, const char *restrict format, /*args */ ..)/
```

- fprintf는 지정한 파일로 형식 지정 방법을 사용하여 출력

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 lseek와 같은 값을 사용
- fseek는 성공하면 0을 실패하면 EOF를 리턴

값	설명
SEEK_SET	파일의 시작 기준
SEEK_CUR	현재 위치 기준
SEEK_END	파일의 끝 기준

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]

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

파일기술자와 파일포인터간 변환

파일기술자와 파일포인터는 상호 변환 가능

파일 포인터 생성: fdopen(3)

- 파일 기술자 -> 파일 포인터

```
#include <stdio.h>
FILE *fdopen(int fildes, const char *mode);
```

파일 기술자 생성: fileno(3)

- 파일 포인터 -> 파일 기술자

```
#include <stdio.h>
int fileno(FILE *stream);
```

fdopen 함수 사용하기

```
01  #include <fcntl.h>
02  #include <stdlib.h>
03  #include <stdio.h>
04
05  int main(void) {
06      FILE *fp;
07      int fd;
08      char str[BUFSIZ];
09
10      fd = open("unix.txt", O_RDONLY);
11      if (fd == -1) {
12          perror("open");
13          exit(1);
14      }
15
16      fp = fdopen(fd, "r");
17
18      fgets(str, BUFSIZ, fp);
19      printf("Read : %s\n", str);
20
21      fclose(fp);
22
23      return 0;
24  }
```

저수준 파일입출력 함수로 파일 오픈

파일 포인터 생성

고수준 파일읽기 함수로 읽기

```
# ex2_18.out
Read : Unix System Programming
```

fileno 함수 사용하기

```
01  #include <unistd.h>
02  #include <fcntl.h>
03  #include <stdlib.h>
04  #include <stdio.h>
05
06  int main(void) {
07      FILE *fp;
08      int fd, n;
09      char str[BUFSIZ];
10
11      fp = fopen("unix.txt", "r");
12      if (fp == NULL) {
13          perror("fopen");
14          exit(1);
15      }
16
17      fd = fileno(fp);
18      printf("fd : %d\n", fd);
19
20      n = read(fd, str, BUFSIZ);
21      str[n] = '\0';
22      printf("Read : %s\n", str);22
23
24      close(fd);
25
26      return 0;
27  }
```

고수준 파일입출력 함수로 파일 오픈

파일 기술자 리턴

저수준 파일읽기 함수로 읽기

```
# ex2_19.out
fd : 3
Read : Unix System Programming
```

요약

파일

- 파일은 관련 있는 데이터들의 집합
- 파일은 장치에 접근하기 위해서도 사용

저수준 파일 입출력과 고수준 파일 입출력

- 저수준 파일 입출력
  - 커널의 시스템 호출을 사용
  - 특수 파일도 읽고 쓸 수 있음 (e.g. /dev/parport)
- 고수준 파일 입출력
  - 표준 입출력 라이브러리
  - 다양한 형태의 파일 입출력 함수를 제공

	저수준 파일 입출력	고수준 파일 입출력
파일 지시자	int fd (파일 기술자)	FILE *fp; (파일 포인터)
특징	• 더 빠르다. • 바이트 단위로 읽고 쓴다. • 특수 파일에 대한 접근이 가능하다.	• 사용하기 쉽다. • 버퍼 단위로 읽고 쓴다. • 데이터의 입출력 동기화가 쉽다. • 여러 가지 형식을 지원한다.

Exercise

Ex.

- /tmp/num.txt 에서 숫자를 읽어서 4의 배수인 경우 four.bin 파일로 저장하고 총 개수를 출력 (4B binary 형태로 저장)
- num.txt 파일 이름은 argument로 받는다

```
./a.out num.txt
1234
```

```
hexdump -d four.bin
0000000   00000   00000   00010   00000   00020   00000   00030   00000
0000010   00040   00000   00050   00000   00060   00000   00070   00000
0000020   00080   00000   00090   00000   00100   00000   00110   00000
0000030   00120   00000   00130   00000   00140   00000   00150   00000
0000040   00160   00000   00170   00000   00180   00000   00190   00000
0000050   00200   00000   00210   00000   00220   00000   00230   00000
0000060   00240   00000   00250   00000   00260   00000   00270   00000
0000070   00280   00000   00290   00000   00300   00000   00310   00000
....
```

Exercise – Structure read/write

다음 구조체를 저장하고 읽어 출력하는 프로그램 작성 (binary 형태)

- 데이터 1000개는 아래와 같이 생성
- fread/fwrite 이용

```
// 구조체 정의
typedef struct {
    int id;
    char name[32];
    int age;
} person;


// 예제 데이터 준비
const int NUM_PEOPLE = 1000;
person *people = malloc(sizeof(person) * NUM_PEOPLE);
if (!people) {
    perror("malloc");
    return 1;
}
for (int i = 0; i < NUM_PEOPLE; i++) {
    people[i].id = i + 1;
    snprintf(people[i].name, sizeof(people[i].name), "person_%d", i + 1);
    people[i].age = 20 + (i % 50);
}
```