4장 C 표준 파일 입출력

4.1 파일 및 파일 포인터

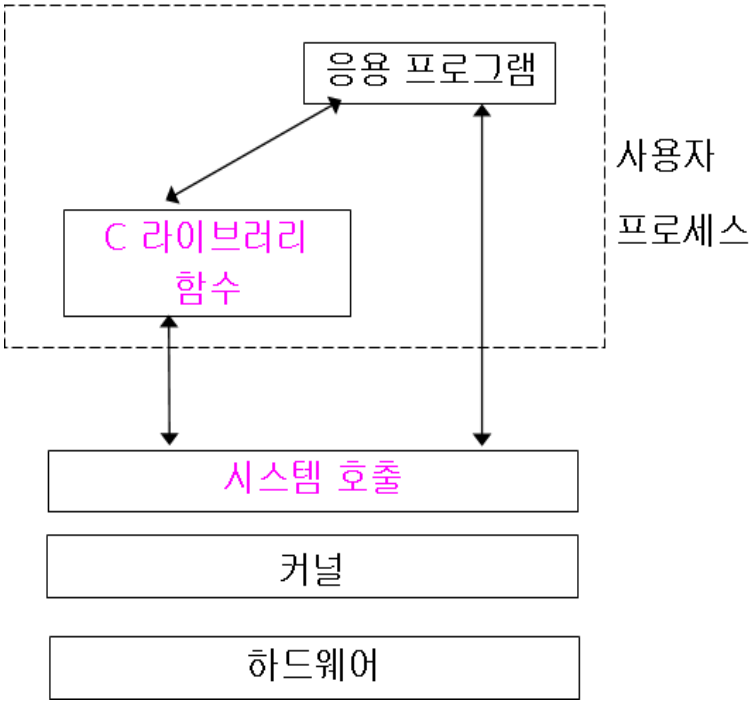
시스템 호출과 C 라이브러리 함수

- 시스템 호출(System Calls)
  - Unix 커널에 서비스 요청하는 호출
  - UNIX man의 Section 2에 설명되어 있음
  - C 함수처럼 호출될 수 있음.

- C 라이브러리 함수(Library Functions)
  - C 라이브러리 함수는 보통 시스템 호출을 포장해 놓은 함수
  - 보통 내부에서 시스템 호출을 함
  - 1975년에 Dennis Ritchie에 의해 작성
  - ANSI C Standard Library

시스템 호출(system call)

- 시스템 호출은 커널에 서비스 요청을 위한 프로그래밍 인터페이스
- 응용 프로그램은 시스템 호출을 통해서 커널에 서비스를 요청한다.



파일

- C 프로그램에서 파일은 왜 필요할까?
  - 변수에 저장된 정보들은 실행이 끝나면 모두 사라진다.
  - 정보를 영속적으로 저장하기 위해서는 파일에 저장해야 한다.

- 유닉스 파일
  - 모든 데이터를 연속된 바이트 형태로 저장한다.

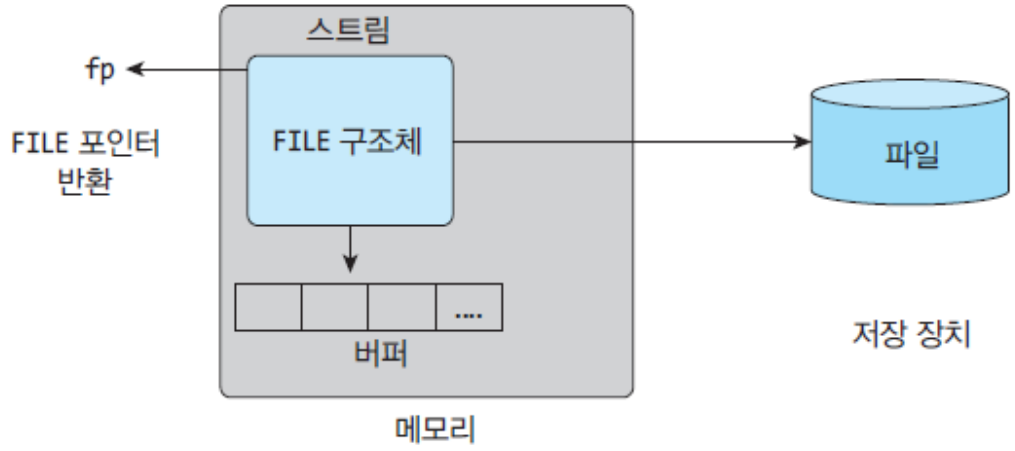
C 언어의 파일 종류

- 텍스트 파일(text file)
  - 사람들이 읽을 수 있는 문자들을 저장하고 있는 파일
  - 텍스트 파일에서 "한 줄의 끝"을 나타내는 표현은 파일이 읽어 들여질 때, C 내부의 방식으로 변환된다.

- 이진 파일(binary file)
  - 모든 데이터는 있는 그대로 바이트의 연속으로 저장
  - 이진 파일을 이용하여 메모리에 저장된 변수 값 형태 그대로 파일에 저장할 수 있다.

파일 입출력

- C 언어의 파일 입출력 과정
  1. 파일 열기: fopen( ) 함수 사용
  2. 파일 입출력 : 다양한 파일 입출력 함수 사용
  3. 파일 닫기: fclose( ) 함수 사용



파일 열기

- 파일을 사용하기 위해서는
  - 반드시 먼저 파일 열기(fopen)를 해야 한다.
  - 파일 열기를 하면 FILE 구조체에 대한 포인터가 리턴되며
  - FILE 포인터는 열린 파일을 나타낸다.

- 함수 fopen()
  - FILE *fopen(const char *filename, const char *mode);
  - const char *filename: 파일명에 대한 포인터
  - const char *mode: 모드로 파일을 여는 형식

- 예 1

```
FILE *fp;
fp = fopen("~/sp/text.txt", "r");
if (fp == NULL) {
    printf("파일 열기 오류₩n");
}
```

- 예 2

```
fp = fopen("outdata.txt", "w");
fp = fopen("outdata.txt", "a");
```

fopen() : 텍스트 파일 열기

모드	의미	파일이 없으면	파일이 있으면
"r"	읽기 전용(read)	NULL 반환	정상 동작
"w"	쓰기 전용(write)	새로 생성	기존 내용 삭제
"a"	추가 쓰기(append)	새로 생성	기존 내용 뒤에 추가
"r+"	읽기와 쓰기	NULL 반환	정상 동작
"w+"	읽기와 쓰기	새로 생성	기존 내용 삭제
"a+"	추가를 위한 읽기와 쓰기	새로 생성	기존 내용 뒤에 추가

FILE 구조체

- 파일 관련 시스템 호출
  - 파일 디스크립터 (file descriptor)

- C 표준 입출력 함수
  - fopen( ) 함수로 파일을 열면 FILE 포인터(FILE *)가 리턴됨
  - 열린 파일을 가리키는 FILE 구조체에 대한 포인터
  - FILE 포인터를 표준 입출력 함수들의 인수로 전달해야 함
  - #include <stdio.h>

- FILE 구조체
  - 하나의 스트림에 대한 정보를 포함하는 구조체
  - 버퍼에 대한 포인터, 버퍼 크기 ...
  - 파일 디스크립터

FILE 구조체

- FILE 구조체: 열린 파일의 현재 상태를 나타내는 필드 변수들
  특히 파일 입출력에 사용되는 버퍼 관련 변수들

```
typedef struct {
    int cnt;                  // 버퍼의 남은 문자 수
    unsigned char*base;  // 버퍼 시작
    unsigned char*ptr;    // 버퍼의 현재 포인터
    unsinged flag;          // 파일 입출력 모드( _IOFBF, _IOLBF, _IONBUF,
                                //       _IOEOF, _IOERR _IOREAD, _IOWRT)
    int fd;                     // 열린 파일 디스크립터
} FILE;// FILE 구조체
```

표준 입력/출력/오류

- 표준 I/O 스트림 (stream)
  - 프로그램이 시작되면 자동으로 open되는 스트림
  - stdin, stdout, stderr
  - FILE*
  - #include <stdio.h>

표준 입출력 포인터	설명	가리키는 장치
stdin	표준 입력에 대한 FILE 포인터	키보드
stdout	표준 출력에 대한 FILE 포인터	모니터
stderr	표준 오류에 대한 FILE 포인터	모니터

파일 닫기

- 파일을 열어서 사용한 후에는 파일을 닫아야 한다.
  - int fclose(FILE *fp);
  - fp는 fopen 함수에서 받았던 포인터
  - 닫기에 성공하면 0, 오류일 때는 EOF(-1)를 리턴한다.

- 예
  - fclose(fp);

4.2 텍스트 파일

파일 입출력 함수

표준 입출력함수	파일 입출력 함수	기능
getchar()	fgetc(), getc()	문자단위로 입력하는 함수
putchar()	fputc(), putc()	문자단위로 출력하는 함수
gets()	fgets()	문자열을 입력하는 함수
puts()	fputs()	문자열을 출력하는 함수
scanf()	fscanf()	자료형에 따라 자료를 입력하는 함수
printf()	fprintf()	자료형에 따라 자료를 출력하는 함수

문자 단위 입출력

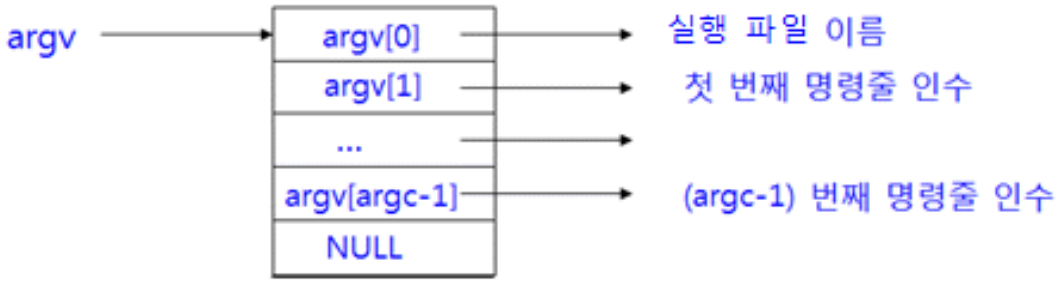
- fgetc() 함수와 fputc() 함수
  - 파일에 문자 단위 입출력을 할 수 있다.

- int fgetc(FILE *fp);
  - fp가 지정한 파일에서 한 문자를 읽어서 리턴한다.
  - 파일 끝에 도달했을 경우에는 EOF(-1)를 리턴한다.

- int fputc(int c, FILE *fp);
  - fp가 가리키는 파일에 한 문자씩 출력하는 함수
  - 리턴값으로 출력하는 문자 리턴
  - 출력시 오류가 발생하면 EOF(-1) 리턴

cat.c

```
#include <stdio.h>
/* 텍스트 파일 내용을 표준출력에 출력 */
int main(int argc, char *argv[])
{
    FILE *fp;
    int c;
    if (argc < 2)
        fp = stdin;                    // 명령줄 인수가 없으면 표준입력 사용
    else  fp = fopen(argv[1],"r");     // 읽기 전용으로 파일 열기

    c = getc(fp);                      // 파일로부터 문자 읽기
    while (c != EOF) {                 // 파일끝이 아니면
        putc(c, stdout);               // 읽은 문자를 표준출력에 출력
        c = getc(fp);                  // 파일로부터 문자 읽기
    }
    fclose(fp);
    return 0;
}
```



copy.c

```
#include <stdio.h>
/* 파일 복사 프로그램 */
int main(int argc, char *argv[])
{
   char c;
   FILE *fp1, *fp2;
   if (argc !=3) {
      fprintf(stderr,  "사용법: %s 파일1 파일2\n", argv[0]);
      return 1;
   }
```

copy.c

```
    fp1 = fopen(argv[1], "r");
    if (fp1 == NULL) {
       fprintf(stderr, "파일 %s 열기 오류\n", argv[1]);
       return 2;
    }

    fp2 = fopen(argv[2], "w");
    while ((c = fgetc(fp1)) != EOF)
       fputc(c, fp2);

    fclose(fp1);
    fclose(fp2);
    return 0;
}
```

기타 파일 관련 함수

- int feof(FILE *fp)
  - 파일 포인터 fp가 파일의 끝을 탐지하면 0이 아닌 값을 리턴하고 파일 끝이면 0을 리턴 한다.

- int ungetc(int c, FILE *p)
  - c에 저장된 문자를 입력 스트림에 반납한다. 마치 문자를 읽지 않은 것처럼 파일 위치 지정자를 1 감소시킨다.

- int fflush(FILE *fp)
  - 아직 기록되지 않고 버퍼에 남아 있는 데이터를 fp가 가리키는 출력 파일에 보낸다. 버퍼 비우기 기능을 수행하는 함수이다.

줄 단위 입출력

- fgets() 함수와 fputs() 함수
  - 텍스트 파일에서 한 줄씩 읽거나 쓸 수 있다.

- char* fgets(char *s, int n, FILE *fp);
  - 파일로부터 한 줄을 읽어서 문자열 포인터 s에 저장하고 s를 리턴
  - 개행문자('₩n')나 EOF를 만날 때까지 파일로부터 최대 n-1 개의 문자를 읽고 읽어온 데이터의 끝에는 널문자('₩0')를 붙여준다.
  - 파일을 읽는 중 파일 끝 혹은 오류가 발생하면 NULL 포인터 리턴.

- int fputs(const char *s, FILE *fp);
  - 문자열 s를 파일 포인터 fp가 가리키는 파일에 출력
  - 성공적으로 출력한 경우에는 출력한 바이트 수를 리턴
  - 출력할 때 오류가 발생하면 EOF 값을 리턴

line.c

```
1 #include <stdio.h>
2 #define MAXLINE 80
3
4 /* 텍스트 파일에 줄 번호 붙여 프린트한다. */
5 int main(int argc, char *argv[])
6 {
7     FILE *fp;
8     int line = 0;
9     char buffer[MAXLINE];
10
11    if (argc != 2) {
12        fprintf(stderr,  "사용법:line 파일이름\n");
13        exit(1);
14    }
```

line.c

```
15
16    if ( (fp = fopen(argv[1],"r")) == NULL)
17    {
18        fprintf(stderr, "파일 열기 오류\n”);
19        exit(2);
20    }
21
22    while (fgets(buffer, MAXLINE, fp) != NULL) { // 한 줄 읽기
23        line++;
24        printf("%3d %s", line, buffer);
25    }
26    exit(0);
27 }
```

포맷 입출력

- fprintf() 함수
  - printf() 함수와 같은 방법으로 파일에 데이터를 출력할 수 있다.
- fscanf() 함수
  - scanf() 함수와 같은 방법으로 파일로부터 데이터를 읽어 들일 수 있다.
- int fprintf(FILE *fp, const char *format, ...);
  - fprintf 함수의 첫 번째 인수 fp는 츨력할 파일에 대한 FILE 포인터
  - 두 번째부터의 인수는 printf 함수와 동일
- int fscanf(FILE *fp, const char *format, ...);
  - fscanf 함수의 첫 번째 인수 fp는 입력받을 파일에 대한 FILE 포인터
  - 두 번째부터의 인수는 scanf 함수와 동일

fprint.c

```
#include <stdio.h>
#include "student.h"
/* 학생 정보를 읽어 텍스트 파일에 저장한다. */
int main(int argc, char* argv[])
{
    struct student rec;
    FILE *fp;
    if (argc != 2) {
        fprintf(stderr, "사용법: %s 파일이름₩n", argv[0]);
        return 1;
    }
    fp = fopen(argv[1], "w");
    printf("%-9s %-7s %-4s₩n", "학번", "이름", "점수");
    while (scanf("%d %s %d", &rec.id, rec.name, &rec.score)==3)
        fprintf(fp, "%d %s %d ", rec.id, rec.name, rec.score);
    fclose(fp);
    return 0;
}
```

fscan.c

```
#include <stdio.h>
#include "student.h"
/* 텍스트 파일에서 학생 정보를 읽어 프린트한다. */
int main(int argc, char* argv[]) {
    struct student rec;
    FILE *fp;
    if (argc != 2) {
        fprintf(stderr, "사용법: %s 파일이름₩n", argv[0]);
        return 1;
    }
    fp = fopen(argv[1], "r");
    printf("%-9s %-7s %-4s₩n", "학번", "이름", "점수");
    while (fscanf(fp,"%d %s %d", &rec.id, rec.name, &rec.score)==3)
        printf("%10d %6s %6d₩n", rec.id, rec.name, rec.score);
    fclose(fp);
    return 0;
}
```

4.3 이진 파일

fopen(): 이진 파일 열기

모드	의미	파일이 없으면	파일이 있으면
"rb"	읽기 전용(read)	NULL 반환	정상 동작
"wb"	쓰기 전용(write)	새로 생성	기존 내용 삭제
"ab"	추가 쓰기(append)	새로 생성	기존 내용 뒤에 추가
"rb+"	읽기와 쓰기	NULL 반환	정상 동작
"wb+"	읽기와 쓰기	새로 생성	기존 내용 삭제
"ab+"	추가를 위한 읽기와 쓰기	새로 생성	기존 내용 뒤에 추가

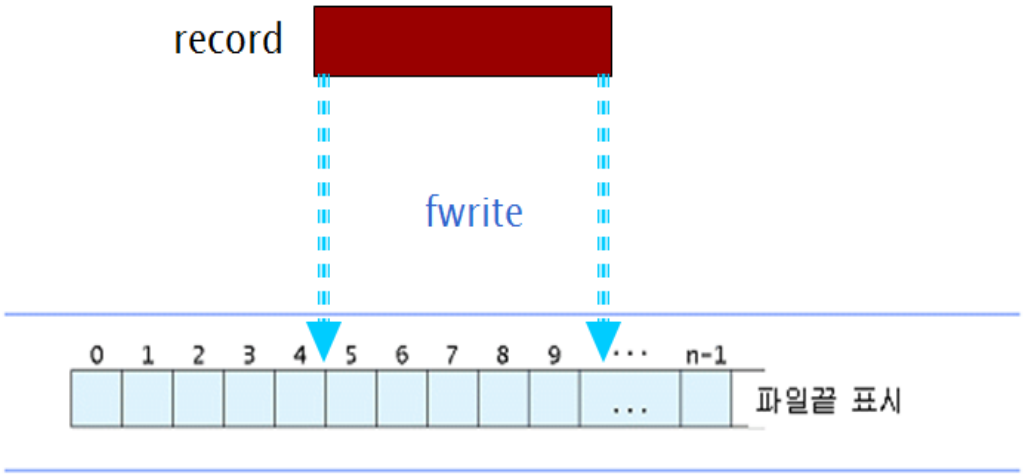
블록 단위 입출력

- fread()와 fwrite()
  - 한번에 일정한 크기의 데이터를 파일에 읽거나 쓰기 위한 입출력 함수

- int fread(void *buf, int size, int n, FILE *fp);
  - fp가 가리키는 파일에서 size 크기의 블록(연속된 바이트)을 n개 읽어서 버퍼 포인터 buf가 가리키는 곳에 저장
  - 읽어온 블록의 개수를 리턴

- int fwrite(const void *buf, int size, int n, FILE *fp);
  - 파일 포인터 fp가 지정한 파일에 버퍼 buf에 저장되어 있는 size 크기의 블록(연속된 바이트)을 n개 기록
  - 성공적으로 출력한 블록 개수를 리턴

fwrite() 함수



블록 입출력

- 기본 아이디어
  - 어떤 자료형의 데이터이던지 그 데이터를 연속된 바이트로 해석해서 파일에 저장
  - 파일에 저장된 데이터를 연속된 바이트 형태로 읽어서 원래 변수에 순서대로 저장하여 원래 데이터를 그대로 복원

- 예: rec 저장

```
struct student rec;
FILE *fp = fopen("stfile", "wb");
...
fwrite(&rec, sizeof(rec), 1, fp);
```

학생 레코드를 파일에 저장하는 예제

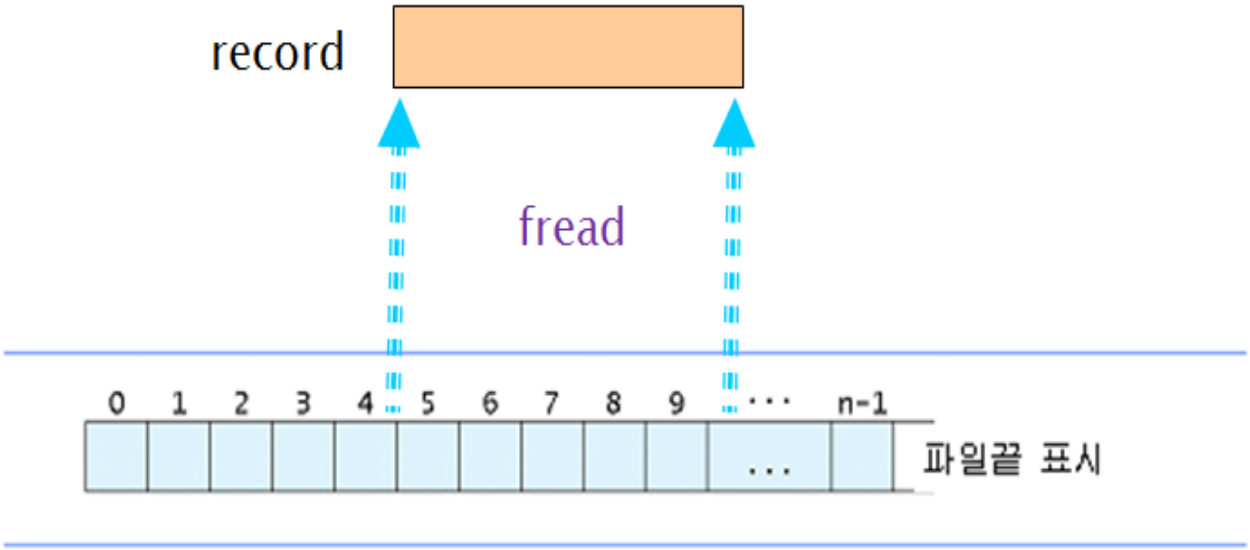
stcreate1. c

```
#include <stdio.h>
#include "student.h"
int main(int argc, char* argv[])
{
    struct student rec;
    FILE *fp;

    if (argc != 2) {
        fprintf(stderr, "사용법: %s 파일이름₩n",argv[0]);
        exit(1);
    }

    fp = fopen(argv[1], "wb");
    printf("%-9s %-7s %-4s₩n", "학번", "이름", "점수");
    while (scanf("%d %s %d", &rec.id, rec.name, &rec.score) == 3)
        fwrite(&rec, sizeof(rec), 1, fp);

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

student.h

```
struct student {
    int id;
    char name[20];
    short score;
};
```

fread() 함수



학생 레코드를 읽어서 출력하는 예제 stprint.c

```
#include <stdio.h>
#include "student.h"
/* 파일에 저장된 모든 학생 정보를 읽어서 출력한다. */
int main(int argc, char* argv[])
{
   struct student rec;
   FILE *fp;
   if (argc != 2) {
      fprintf(stderr, "사용법: %s 파일이름\n", argv[0]);
      return 1;
   }
   if ((fp = fopen(argv[1], "rb")) == NULL ) {
      fprintf(stderr, "파일 열기 오류\n");
      return 2;
   }
```

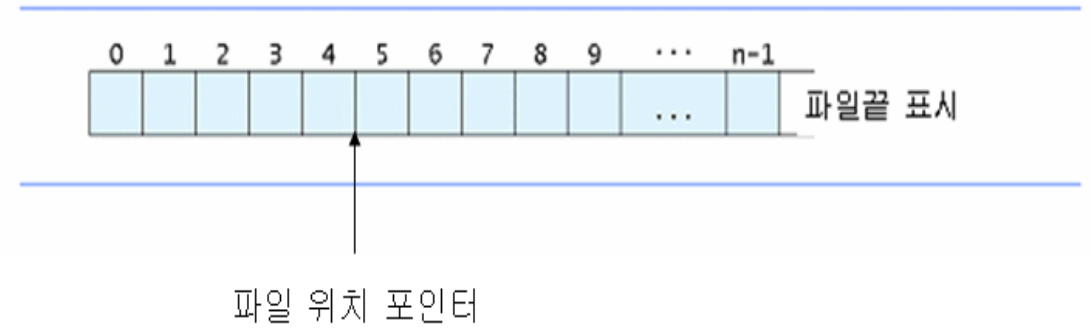
stprint.c

```
    printf("--------------------------------\n");
    printf("%10s %6s %6s\n", "학번", "이름", "점수");
    printf("--------------------------------\n");

    while (fread(&rec, sizeof(rec), 1, fp) > 0)
       if (rec.id != 0)
          printf("%10d %6s %6d\n", rec.id,  rec.name,  rec.score);

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

4.4 임의 접근

파일 내 위치

- 현재 파일 위치(current file position)
  - 열린 파일에서 다음 읽거나 기록할 파일 내 위치
- 파일 위치 포인터(file position pointer)
  - 시스템 내에 그 파일의 현재 파일 위치를 저장하고 있다.

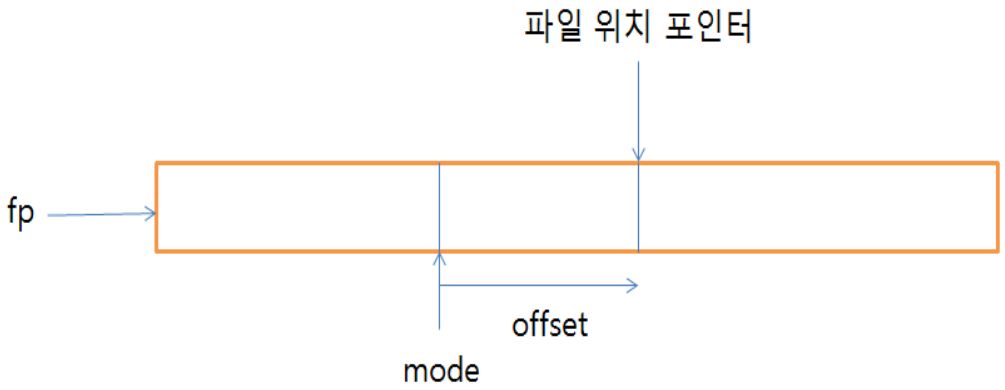


파일 위치 관련 함수

- fseek(FILE *fp, long offset, int mode)
  - 파일 위치 포인터를 임의로 설정할 수 있는 함수
- rewind(FILE *fp)
  - 현재 파일 위치를 파일 시작에 위치시킴.
- ftell(FILE *fp)
  - 파일의 현재 파일 위치를 나타내는 파일 위치 지정자 값 리턴

임의 접근 함수 fseek()

- fseek(FILE *fp, long offset, int mode)
  - FILE 포인터 fp가 가리키고 파일의 파일 위치를
  - 모드(mode) 기준으로 오프셋(offset)만큼 옮긴다.



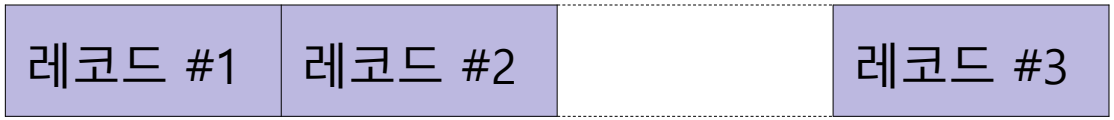
기호	값	의미
SEEK_SET	0	파일 시작
SEEK_CUR	1	현재 위치
SEEK_END	2	파일 끝

fseek() 함수

- 파일 위치 이동
  - fseek(fp, 0L, SEEK_SET); 파일 시작으로 이동(rewind)
  - fseek(fp, 100L, SEEK_SET); 파일 시작에서 100바이트 위치로
  - fseek(fp, 0L, SEEK_END); 파일 끝으로 이동(append)
- 레코드 단위로 이동
  - fseek(fp, n * sizeof(record), SEEK_SET); n+1번째 레코드 시작위치로
  - fseek(fp, sizeof(record), SEEK_CUR); 다음 레코드 시작위치로
  - fseek(fp, -sizeof(record), SEEK_CUR); 전 레코드 시작위치로
- 파일끝 이후로 이동
  - fseek(fp, sizeof(record), SEEK_END); 파일끝에서 한 레코드 다음 위치로 이동

파일끝 이후로 이동:예

```
fwrite(&record1, sizeof(record), 1, fp);
fwrite(&record2, sizeof(record), 1, fp);
fseek(fp, sizeof(record), SEEK_END);
fwrite(&record3, sizeof(record), 1, fp);
```



임의 접근을 이용한 학생 레코드 저장 stcreate2.c

```
#include <stdio.h>
#include "student.h"
#define START_ID 1001001
/* 구조체를 이용하여 학생 정보를 파일에 저장한다. */
int main(int argc, char* argv[])
{
   struct student rec;
   FILE *fp;
   if (argc != 2) {
      fprintf(stderr, "사용법: %s 파일이름\n", argv[0]);
      exit(1);
   }
```

stcreate2.c

```
    fp = fopen(argv[1], "wb");
    printf("%7s %6s %4s\n", "학번", "이름", "점수");
    while (scanf("%d %s %d", &rec.id,    rec.name, &rec.score) == 3) {
       fseek(fp,  (rec.id – START_ID)* sizeof(rec),  SEEK_SET);
       fwrite(&rec, sizeof(rec), 1, fp);
    }
    fclose(fp);
    exit(0);
}
```

임의 접근을 이용한 학생 레코드 검색 stquery.c

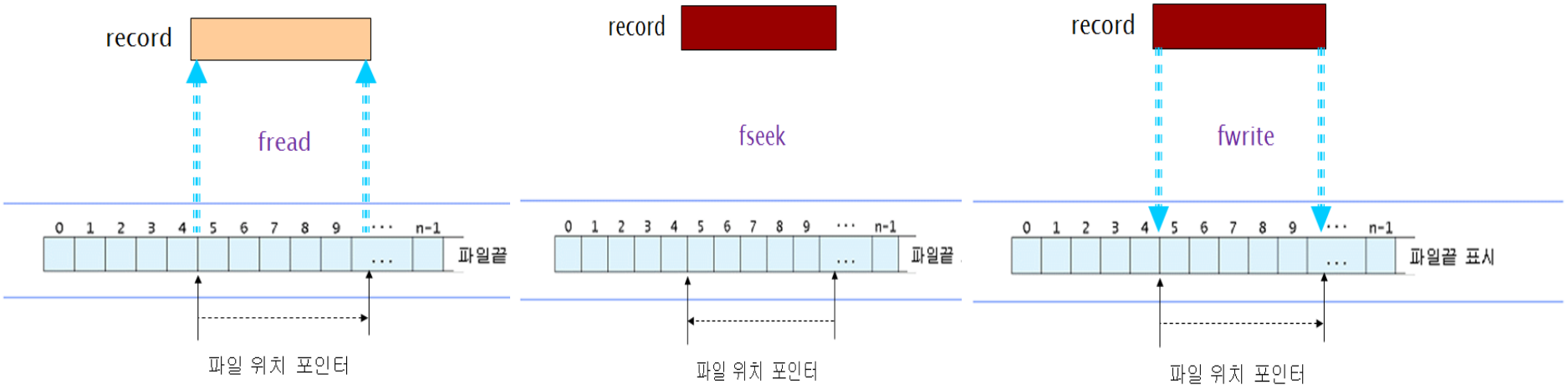
```
#include <stdio.h>
#include "student.h"
int main(int argc, char *argv[])
{
   struct student rec;
   char c;
   int id;
   FILE *fp;
   if (argc != 2) {
      fprintf(stderr, "사용법: %s 파일이름\n", argv[0]);
      exit(1);
   }

   if ((fp = fopen(argv[1], "rb")) == NULL ) {
      fprintf(stderr, "파일 열기 오류\n");
      exit(2);
   }
```

stquery.c

```
    do {
      printf("검색할 학생의 학번 입력: ");
      if (scanf("%d", &id) == 1) {
        fseek(fp, (id - START_ID) *sizeof(rec), SEEK_SET);
        if ((fread(&rec, sizeof(rec), 1, fp) > 0)  &&  (rec.id != 0))
          printf("학번: %8d 이름: %4s 점수: %4d\n", rec.id, rec.name, rec.score);
         else printf("레코드 %d 없음\n", id);
      }
      else printf("입력 오류");

      printf("계속하겠습니까?(Y/N)");
      scanf(" %c", &c);
    } while (c == 'Y');

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

레코드 수정 과정

(1) 파일로부터 해당 레코드를 읽어서
(2) 이 레코드를 수정한 후에
(3) 수정된 레코드를 다시 파일 내의 원래 위치에 써야 한다.



임의 접근을 이용한 학생 레코드 수정 stupdate.c

```
#include <stdio.h>
#include "student.h"
int main(int argc, char *argv[])
{
   struct student rec;
   int id;
   char c;
   FILE *fp;

   if (argc != 2) {
      fprintf(stderr, "사용법: %s 파일이름\n", argv[0]);
      exit(1);
   }

   if ((fp = fopen(argv[1], "rb+")) == NULL ) {
      fprintf(stderr, "파일 열기 오류\n");
      exit(2);
   }
```

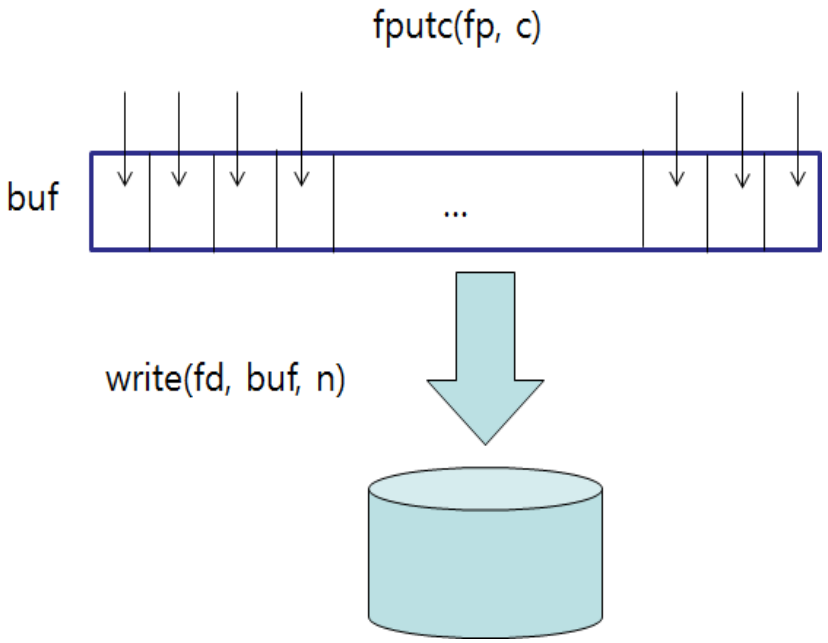
stupdate.c

```
do {
    printf("수정할 학생의 학번 입력: ");
    if (scanf("%d", &id) == 1) {
        fseek(fp, (id - START_ID) * sizeof(rec), SEEK_SET);
        if ((fread(&rec, sizeof(rec), 1, fp) > 0)&&(rec.id != 0)) {
            printf("학번: %8d 이름: %4s 점수: %4d₩n", rec.id, rec.name, rec.score);
            printf("새로운 점수 입력: ");
            scanf("%d", &rec.score);
            fseek(fp, -sizeof(rec), SEEK_CUR);
            fwrite(&rec, sizeof(rec), 1, fp);
        } else printf("레코드 %d 없음₩n", id);
    } else printf("입력오류₩n");

    printf("계속하겠습니까?(Y/N)");
    scanf(" %c",&c);
} while (c == 'Y');
fclose(fp);
exit(0);
}
```

4.5 버퍼 입출력

C 라이브러리 버퍼

- C 라이브러리 버퍼 사용 목적
  - 디스크 I/O 수행의 최소화
    - read (), write () 함수 호출의 최소화
  - 최적의 크기 단위로 I/O 수행
  - 시스템 성능 향상



- C 라이브러리 버퍼 방식
  - 완전 버퍼(fully buffered) 방식
  - 줄 버퍼(line buffered) 방식
  - 버퍼 미사용(unbuffered) 방식

C 라이브러리 버퍼 방식

- 완전 버퍼 방식
  - 버퍼가 꽉 찼을 때 실제 I/O 수행
  - 디스크 파일 입출력
- 줄 버퍼 방식
  - 줄 바꿈 문자(newline)에서 실제 I/O 수행
  - 터미널 입출력 (stdin, stdout)
- 버퍼 미사용 방식
  - 버퍼를 사용하지 않는다.
  - 표준 에러 (stderr)

buffer.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define _IO_UNBUFFERED 0x0002
#define _IO_LINE_BUF 0x0200

int main(int argc, char *argv[])
{
    FILE *fp;
    if (!strcmp(argv[1], "stdin")) {
        fp = stdin;
        printf("한 글자 입력:");
        if (getchar() == EOF) perror("getchar");
    }
    else if (!strcmp(argv[1], "stdout"))
        fp = stdout;
    else if (!strcmp(argv[1], "stderr"))
        fp = stderr;
```

buffer.c

```
    else if ((fp = fopen(argv[1], "r")) == NULL) {
        perror("fopen");
        exit(1);
    }
    else if (getc(fp) == EOF) perror("getc");

    printf("스트림 = %s, ", argv[1]);

    if (fp->_flags & _IO_UNBUFFERED)
        printf("버퍼 미사용");
    else if (fp->_flags & _IO_LINE_BUF)
        printf("줄 버퍼 사용");
    else
        printf("완전 버퍼 사용");

    printf(", 버퍼 크기 = %d₩n", fp->_IO_buf_end - fp->_IO_buf_base);
    exit(0);
}
```

버퍼 방식 조사

$ buffer stdin
한 글자 입력:a
스트림 = stdin, 줄 버퍼 사용, 버퍼 크기 = 1024
$ buffer stdout
스트림 = stdout, 줄 버퍼 사용, 버퍼 크기 = 1024
$ buffer stderr
스트림 = stderr, 버퍼 미사용, 버퍼 크기 = 0
$ buffer text
스트림 = text, 완전 버퍼 사용, 버퍼 크기 = 4096

setbuf()/setvbuf()

```
#include <stdio.h>

void   setbuf (FILE *fp, char *buf );
int    setvbuf (FILE *fp, char *buf,  int mode, size_t size );
```

- 버퍼의 관리 방법을 변경한다
- 호출 시기
  - 스트림이 오픈된 후,
  - 입출력 연산 수행 전에 호출되어야 함

setbuf()

void setbuf (FILE *fp, char *buf);

- 버퍼 사용을 on/off 할 수 있다.

- buf 가 NULL 이면 버퍼 미사용 방식
- buf 가 BUFSIZ 크기의 공간을 가리키면 완전/줄 버퍼 방식
  - 터미널 장치면 줄 버퍼 방식
  - 그렇지 않으면 완전 버퍼 방식

setvbuf()

int setvbuf (FILE *fp, char *buf, int mode, size_t size);

- 버퍼 사용 방법을 변경
- 리턴 값: 성공하면 0, 실패하면 nonzero
- mode
  - _IOFBF : 완전 버퍼 방식
  - _IOLBF : 줄 버퍼 방식
  - _IONBF : 버퍼 미사용 방식

setvbuf()

- mode == _IONBF
  - *buf* 와 *size* 는 무시됨
- mode == _IOLBF or _IOFBF
  - buf 가 NULL이 아니면
    - buf 에서 *size* 만큼의 공간 사용
  - NULL이면 라이브러리가 알아서 적당한 크기 할당 사용
    - stat 구조체의 st_blksize 크기 할당 (disk files)
    - st_blksize 값을 알 수 없으면 BUFSIZ 크기 할당 (pipes)

예제:setbuf

```
#include <stdio.h>

main()
{
  printf("안녕하세요, "); sleep(1);
  printf("리눅스입니다!"); sleep(1);
  printf(" ₩n"); sleep(1);

  setbuf(stdout, NULL);
  printf("여러분, "); sleep(1);
  printf("반갑습니다"); sleep(1);
  printf(" ^^"); sleep(1);
  printf(" ₩n"); sleep(1);
}

$ setbuf
안녕하세요, 리눅스입니다!

여러분, 반갑습니다 ^^
```

예제:setvbuf

```
#include <stdio.h>

int main( void )
{
    char buf[1024];
    FILE *fp1, *fp2;

    fp1 = fopen("data1", "a");
    fp2 = fopen("data2", "w");

    if( setvbuf(fp1, buf, _IOFBF, sizeof( buf ) ) != 0 )
        printf("첫 번째 스트림: 잘못된 버퍼₩n" );
    else
        printf("첫 번째 스트림: 1024 바이트 크기 버퍼 사용₩n" );

    if( setvbuf(fp2, NULL, _IONBF, 0 ) != 0 )
        printf("두 번째 스트림: 잘못된 버퍼₩n" );
    else
        printf("두 번째 스트림: 버퍼 미사용₩n" );
}
```

```
$ setvbuf
첫 번째 스트림: 1024 바이트 크기 버퍼 사용
두 번째 스트림: 버퍼 미사용
```

fflush()

```
#include <stdio.h>

int  fflush (FILE *fp);
```

- *fp* 스트림의 출력 버퍼에 남아있는 내용을 write() 시스템 호출을 통하여 커널에 전달한다
- 리턴 값 : 성공하면 0, 실패하면 EOF (-1)
- *fp* 가 NULL이면, 모든 출력 스트림의 출력 버퍼에 남아있는 내용을 커널에 전달한다

4.6 기타 함수

문자열 처리 함수

- #include <string.h>
- char *strcpy(char *s, const char *t);
  - 널문자를 포함해서 문자열 t를 문자열 s에 복사하고 s를 반환한다.
- char *strncpy(char *s, const char *t, size_t n);
  - 최대 n개 문자까지 복사
- char *strcat(char *s, const char *t);
  - 문자열 t를 문자열 s에 접합하고 s를 반환한다.
- char *strncat(char *s, const char *t, size_t n);
  - 최대 n개 문자까지 문자 접합
- int strcmp(const char *s, const char *t);
  - 문자열 s를 문자열 t와 비교한다. s<t이면 음수 값, s==t이면 0, s>t이면 양수 값을 반환한다.
- int strncmp(const char *s, const char *t, size_t n);
  - 최대 n개 문자까지 비교

문자열 처리 함수

없으면 NULL 반환

- char *strchr(const char *s, int c);
  - 문자열 s 내에서 처음 나타난 문자 c에 대한 포인터를 반환한다.
- char *strrchr(const char *s, int c);
  - 문자열 s 내에서 마지막으로 나타난 문자 c에 대한 포인터를 반환한다.
- char *strpbrk(const char *s, const char *t);
  - 문자열 s 내에서 문자열 t 내의 문자가 처음 나타난 곳에 대한 포인터를 반환.
- char *strstr(const char *s, const char *t);
  - 문자열 s 내에서 부분문자열 t가 처음 나타난 곳에 대한 포인터를 반환한다.
- size_t strlen(const char *s);
  - 문자열 s의 길이를 반환한다.
- char *strtok(char *s, const char *t);
  - 문자열 t 내의 문자들을 구분자로 사용하여 문자열 s를 토큰들로 나누어 준다
  - 첫 호출할 때 문자열 s를 주면 첫 번째 토큰을 구해서 문자열로 반환하고
  - 다음 호출부터는 첫 번째 인자로 NULL을 주면 다음 토큰을 반환해준다.

핵심 개념

- 파일은 모든 데이터를 연속된 바이트 형태로 저장한다.
- 파일을 사용하기 위해서는 반드시 파일 열기 fopen()를 먼저 해야 하며 파일 열기를 하면 FILE 구조체에 대한 포인터가 리턴된다.
- FILE 포인터는 열린 파일을 나타낸다.
- fgetc() 함수와 fputc() 함수를 사용하여 파일에 문자 단위 입출력을 할 수 있다.
- fgets() 함수와 fputs() 함수를 이용하여 텍스트 파일에서 한 줄씩 읽거나 쓸 수 있다.
- fread()와 fwrite() 함수는 한 번에 일정한 크기의 데이터를 파일에 읽거나 쓴다.
- 열린 파일에서 다음 읽거나 쓸 파일 내 위치를 현재 파일 위치라고 하며 파일 위치 포인터가 그 파일의 현재 파일 위치를 가리키고 있다.
- fseek() 함수는 현재 파일 위치를 지정한 위치로 이동시킨다.