리눅스 프로그래밍

C 표준 파일 입출력

2025.09.19. Fri.

전북대학교 컴퓨터인공지능학부
박순찬 교수

전북대학교
JEONBUK NATIONAL UNIVERSITY

4장 강의 목차

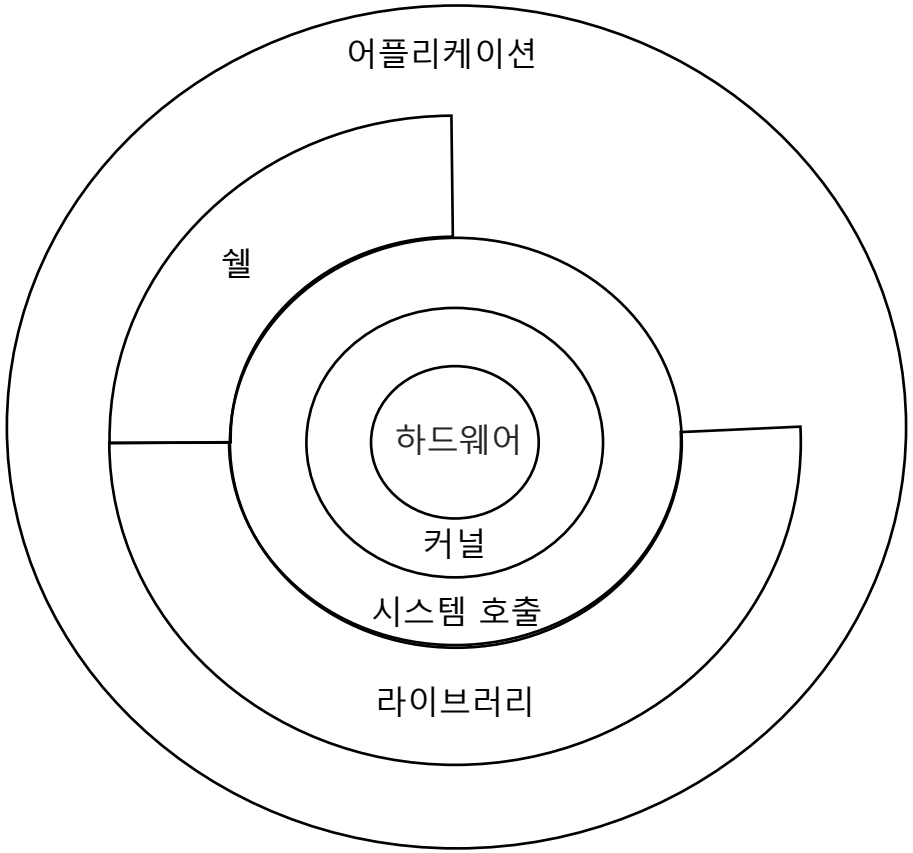
- 파일 및 파일 포인터
- 텍스트 파일
- 이진 파일
- 임의 접근
- 기타 함수

4장 강의 목차

- 파일 및 파일 포인터
- 텍스트 파일
- 이진 파일
- 임의 접근
- 기타 함수

(Recap.) 유닉스 시스템 구조

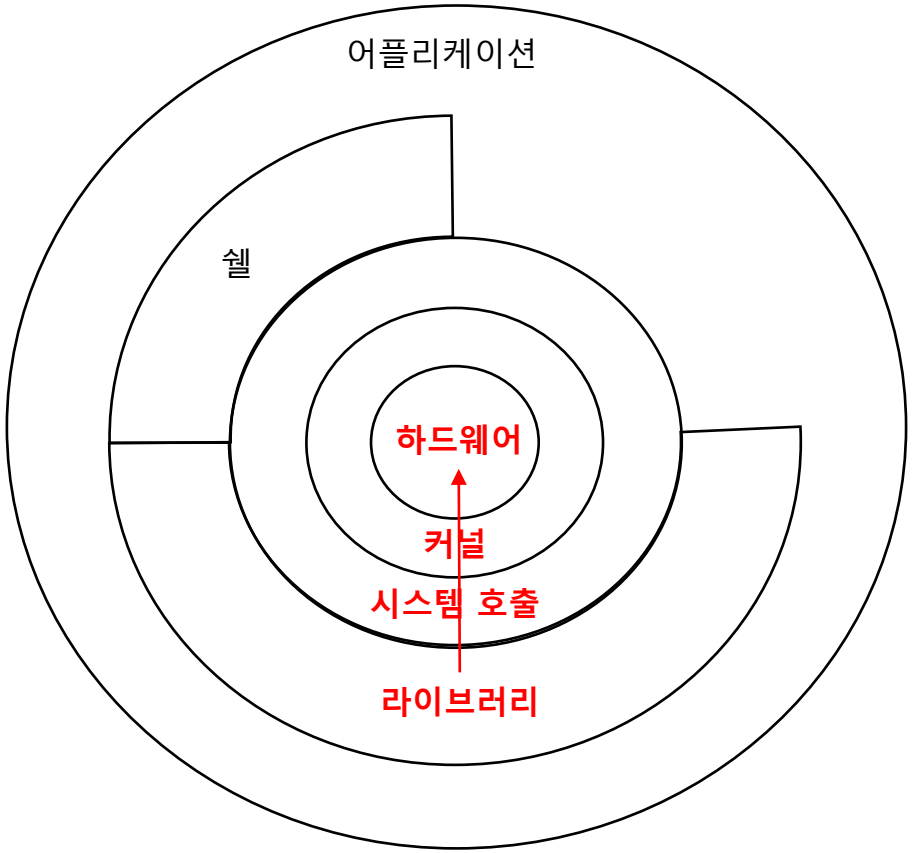
- 커널 (Kernel)
  - 운영체제의 핵심으로 하드웨어 운영/관리
- 시스템 호출 (System call)
  - 유닉스 커널에 대한 프로그래밍 인터페이스
- 쉘 (Shell)
  - 사용자의 명령어 입력에 대한 해석기



시스템 호출과 C 라이브러리 함수

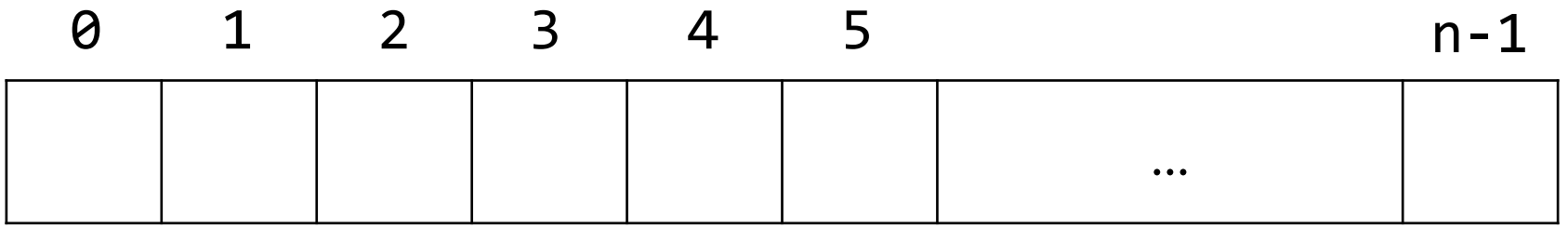
- 시스템 호출 (System Calls)
  - 유닉스 커널에 서비스 요청하는 호출
  - C함수처럼 호출될 수 있음

- C 라이브러리 함수 (Library Functions)
  - C 라이브러리 함수는 보통 시스템 호출을 포장해 놓은 함수
  - 보통 내부에 시스템 호출을 포함하고 있음



파일

- C 프로그램에서 파일은 왜 필요할까?
  - 변수에 저장된 정보들은 실행이 끝나면 모두 사라짐
  - 정보를 저장하기 위해서는 물리적인 공간에 파일 형태로 저장해야 함
- 유닉스 파일
  - 모든 데이터를 연속된 바이트 형태로 저장

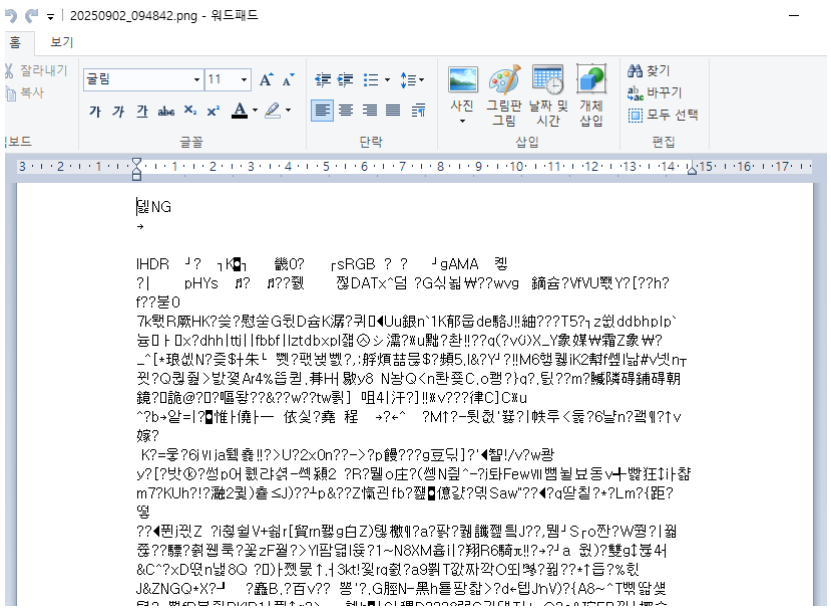


C의 파일

C 언어의 파일 종류

- 텍스트 파일 (text file)
  - 사람이 읽을 수 있는 '문자' 형태로 저장되어 읽고 수정 가능함
  - 값의 경우 숫자를 문자열로 바꾼 다음에 저장해야 함 (`int` → `char*`)
  - 여러 개의 줄로 이루어질 수 있으며 줄의 끝을 나타내는 "₩n" 를 포함하는 것이 특징
  - 텍스트 에디터로 열어서 내용 확인 가능
  - 텍스트 파일의 예: 로그 파일, CSV, JSON 등

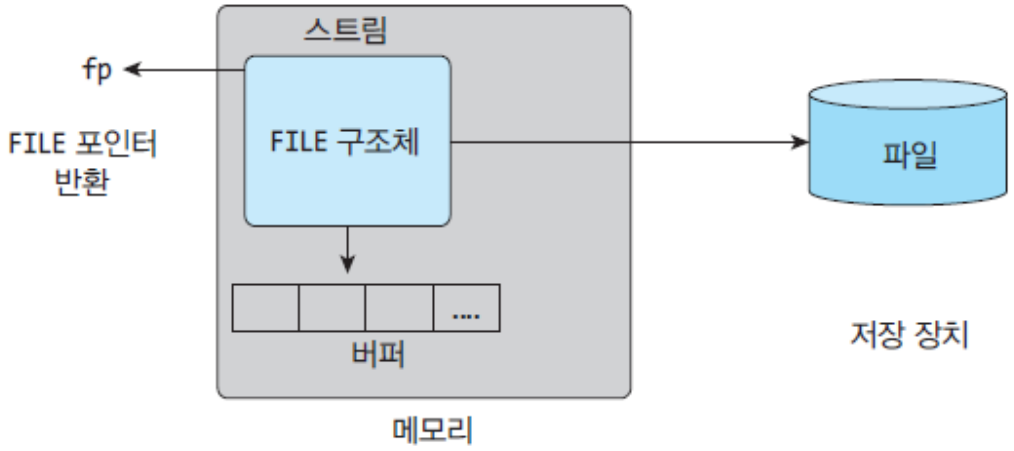
- 이진 파일 (binary file)
  - 모든 데이터를 있는 그대로 바이트의 연속으로 저장
  - 메모리에 저장된 변수 값 형태 그대로 `char` 변환 없이 파일 저장 가능
  - 사람이 실제로 열어서 읽거나 수정하기 어려움
  - 예) 이미지, 오디오, 영상, 등



이미지 파일 .png를 워드패드로 열면 깨진 정보들이 표시된다

파일 입출력

- C언어의 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

- C에서의 파일 열기
  - 파일을 사용하기 위해서는 반드시 열기 (fopen)이 선행되어야 함
  - fopen은 FILE 구조체에 대한 포인터를 리턴하며 이는 열린 파일을 나타냄

- 함수 fopen()

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

const char *filename : 파일명에 대한 문자열
const char *mode : 읽기/쓰기 모드에 대한 정의

fopen()에서의 mode

- 텍스트 파일 열기 관련 mode

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

- FILE 구조체 안에는 열린 파일을 위한 여러 필드 변수들이 선언되어 있음
  - stdio.h 에 선언되어 있음
  - 입출력 최적화를 위해서 버퍼가 존재하고 일정크기가 되면 한꺼번에 보내는 역할을 함

```
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
```

스트림

- 표준 I/O 스트림 (stream)
  - C 프로그램이 시작되면 자동으로 열리고, 종료되면 닫히는 스트림
  - 프로그램의 입력, 출력, 오류 메시지를 분리해서 전달하고 받기 위한 일종의 통로

표준 입출력 포인터	설명	가리키는 장치	대표 함수
`stdin`	표준 입력에 대한 FILE 포인터	키보드	`scanf(), fgets()`
`stdout`	표준 출력에 대한 FILE 포인터	모니터	`printf(), puts()`
`stderr`	표준 오류에 대한 FILE 포인터	모니터	`fprintf(stderr, ...)`

파일 닫기

- 파일을 열어서 사용(읽기/쓰기)한 후에는 파일을 닫아야 함
  - 입력 인자: FILE 구조체 (fopen의 출력임)
  - 출력 인자: 파일 닫기 성공 여부. 성공:0, 오류:-1

```
int fclose(FILE *fp);
```

FILE 구조체 fp에 해당하는 파일을 닫고 성공하면 0, 오류면 -1를 출력함

- 별도의 닫기 없이도 프로그램 종료 시 열려 있는 파일은 닫게 되어 있음
  - 하지만 한 번에 열 수 있는 파일 수 제한이 있기 때문에 불필요한 파일은 바로바로 닫는 것이 효과적

파일 열기/닫기 예제

- 예제 코드
  1. `input.txt` 파일을 `r` 모드로 오픈
  2. `output.txt` 파일을 `w` 모드로 오픈
  3. 입력 파일을 한 줄씩 내려가며 데이터 확인
  4. “Hello”라는 단어가 있을 시 출력파일에 씀
  5. 모든 줄 확인 후 파일 닫고 종료

- 함수 참고
  - `fgets` : size크기만큼 읽음
  - `strstr` : 두 스트링이 같은 지 확인

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

int main() {
    FILE *fin, *fout;
    char line[256];  // 한 줄씩 읽을 버퍼

    // 파일 열기
    fin = fopen("input.txt", "r");       // 텍스트 모드로 읽기
    fout = fopen("output.txt", "w");     // 텍스트 모드로 쓰기

    if (fin == NULL || fout == NULL) {
        fprintf(stderr, "파일 열기에 실패했습니다.\n");
        return 1;
    }

    // 입력 파일을 한 줄씩 읽음
    while (fgets(line, sizeof(line), fin) != NULL) {
        // "Hello"라는 단어가 있는 줄만 출력 파일에 씀
        if (strstr(line, "Hello") != NULL) {
            fputs(line, fout);
        }
    }

    // 파일 닫기
    fclose(fin);
    fclose(fout);

    printf("필터링된 내용이 output.txt에 저장되었습니다.\n");
    return 0;
}
```

4장 강의 목차

- 파일 및 파일 포인터
- 텍스트 파일
- 이진 파일
- 임의 접근
- 버퍼 입출력
- 기타 함수

파일 입출력 함수

- C언어는 아래와 같은 함수들을 텍스트 파일 입출력을 위해서 제공함
  - 이해하면 좋을 키워드
    - 함수명에 (get/put), 함수명 뒤에 붙는 (c/s), (함수명 앞 f/함수명 뒤 f)

파일 입출력 함수	기능
`fgetc(), getc()`	From file, 문자단위로 읽는 함수
`fputc(), putc()`	To file, 문자단위로 쓰는 함수
`fgets()`	From file, 문자열을 읽는 함수
`fputs()`	To file, 문자열을 쓰는 함수
`fscanf()`	From file, 포맷에 따라 자료를 읽는 함수
`fprintf()`	To file, 포맷에 따라 자료를 쓰는 함수

문자 단위 입출력

- **`int fgetc(FILE *fp)`**
  - fp가 지정한 파일에서 한 문자를 읽고 포인터 이동
  - 파일의 끝에 도달했을 경우에는 EOF (-1)를 리턴함

- **`int fputc(int c, FILE *fp)`**
  - fp가 가리키는 파일에 한 문자씩 쓰고, 포인터 이동하고, 쓰어진 문자를 반환
  - 쓰기 시 오류가 발생하면 EOF (-1) 리턴함

명령줄 인수 사용법

- C프로그램은 프로그램을 실행할 때 명령줄 인수를 받을 수 있음
- 명령줄 인수 리스트를 나타내는 포인터 배열 argv의 구성
  - 이를 활용하여 프로그램 실행 시 필요한 인수들을 받을 수 있음
  - `int main(int argc, char *argv[]){` … 와 같이 main함수에 설정
  - 그 이후 [명령어] [인수1] [인수2] …로 실행 가능
    - 예) `catc linux_history.txt`

`argv[0]`	`argv[1]`	`argv[2]`	…	`argv[argc-1]`

파일 내용 출력 예제

- 리눅스 cat 명령어를 c로 구현해보자
- (L09): 명령줄 인수의 개수를 확인하고 인수가 없으면 표준입력, 있으면 파일형태로 읽도록 함
  - (L10): 표준 입력을 사용
  - (L12): 인수를 이용하여 파일 형태로 읽음
- (L14~17): `while`문
  - getc(fp) 호출로 문자를 읽음
  - 읽은 문자를 `putc(c, stdout)` 로 출력
  - 문자가 파일 끝(`EOF`)를 만날 때 까지 반복

program 4.1 : catc.c

```c
#include <stdio.h>

/* Print the contents of a text file to standard output */
int main(int argc, char *argv[])
{
    FILE *fp;
    int c;

    if (argc < 2)
        fp = stdin;                 // Use standard input if no command-line argument
    else
        fp = fopen(argv[1], "r");   // Open file in read mode

    c = getc(fp);                   // Read a character from file
    while (c != EOF) {              // While not end of file
        putc(c, stdout);            // Write the character to standard output
        c = getc(fp);               // Read next character from file
    }

    fclose(fp);                     // Close file
    return 0;
}
```

글자단위 입출력을 활용한 파일 복사 예제 실습

- Complete the code and execute to copy [file1] to [file2]
  - command
    - `$./copy [file1] [file2]`
    - `$./copy copy.c copy_tmp.c`
  - read and write one character by one character
  - As a result, this code will be identical with Linux command “cp”
- Functions to use
  - `fopen()`
  - `fgetc()`
  - `fputc()`

program 4.2 : copy.c

```c
int main(int argc, char *argv[])
{
    char c;
    FILE *fp1, *fp2;

    if (argc != 3) {
        fprintf(stderr, "Usage: %s file1 file2\n", argv[0]);
        return 1;
    }

    //* [DO] read a file with the first argument with "read-only" mode (hint: fp1 = fopen(...))
    if (fp1 == NULL) {
        fprintf(stderr, "Error opening file %s\n", argv[1]);
        return 2;
    }

    //* [DO] read a file with the second argument with "write-only" mode (hint: fp2 = fopen(...))
    //* [DO] using while loop, read a character from fp1 until EOF, and write the character to fp2.

    while (/*[DO](hit: fgetc()*/)
        /*[DO]:(hint: fputc()*/

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

```

줄 단위 입출력

- **`char* fgets(char *s, int n, FILE *fp)`**
  - 파일로부터 한 줄을 읽어와서 문자열 포인터 s에 저장하고 s를 리턴함
  - 줄바꿈 문자('₩n')나 데이터의 끝에는 NULL 문자 ('₩0') 붙임
  - 파일 읽는 중 파일 끝 혹은 오류가 발생하면 NULL 리턴함

- **`int fputs(const char *s, FILE *fp)`**
  - 문자열 s를 파일 포인터fp가 가리키는 파일에 출력
  - 성공적으로 출력한 경우에는 출력한 바이트 수를 리턴, 오류 발생시 EOF(-1) 리턴

줄 단위 입출력 예제

- (L11-14) 인수 확인
- (L16-19) 인수 파일 읽기 (+예외처리)
- (L21~24) fgets를 이용한 한 줄 읽기

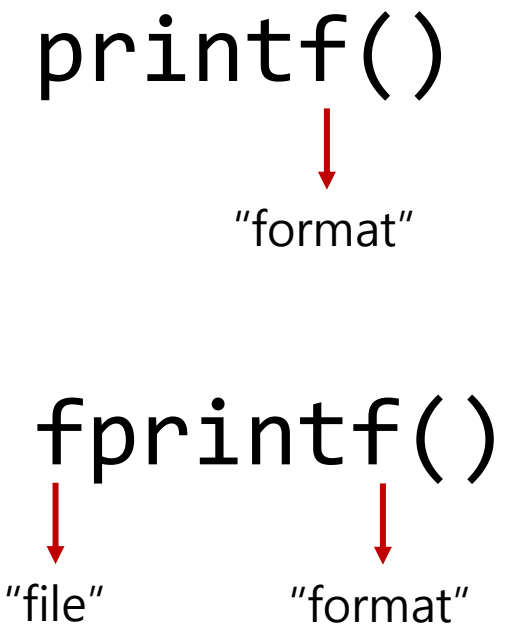
```c
#include <stdio.h>
#define MAXLINE 80

/* 텍스트 파일에 줄 번호 붙여 프린트한다. */
int main(int argc, char *argv[])
{
    FILE *fp;
    int line = 0;
    char buffer[MAXLINE];

   if (argc != 2){
       fprintf(stderr,  "사용법:line 파일이름\n");
       return -1;
   }

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

   while (fgets(buffer, MAXLINE, fp) != NULL) { // 한 줄 읽기
       line++;
       printf("%3d %s", line, buffer);
   }
   return 0;
}
```

포맷 입출력

- fprintf(): printf() 함수와 같은 방법으로 파일에 데이터 출력
- fscanf() : scanf() 함수와 같은 방법으로 파일로부터 데이터 로드

- **int fprintf(FILE *fp, const char *format, ...)**
  - fprintf 함수의 첫 번째 인수 fp는 출력할 파일에 대한 포인터
  - 두 번째 이하 인수는 printf 함수와 동일

- **int fscanf(FILE *fp, const char *format, ...)**
  - fscanf 함수의 첫 번째 인수 fp는 출력할 파일에 대한 포인터
  - 두 번째 이하 인수는 scanf 함수와 동일



포맷 입출력 예제 1

- 키보드로 학생 정보 입력받아 파일로 저장
  - (L02, L07) student 헤더에 정의된 구조체 선언
  - (L15) 인수로 받은 파일을 쓰기 전용으로 열기
  - (L18) 키보드로부터 `scanf` 함수를 이용하여 학번 이름 점수를 입력 받음 (띄워쓰기)
  - **(L19) `fprintf` 함수를 이용하여 파일에 저장**

`student.h`

```
1 struct student {
2     int id;
3     char name[20];
4     short score;
5 };
```

`fprint.c`

```
1 #include <stdio.h>
2 #include "student.h"
3
4 /* 학생 정보를 읽어 텍스트 파일에 저장한다. */
5 int main(int argc, char* argv[])
6 {
7     struct student rec;
8     FILE *fp;
9
10    if (argc != 2) {
11        fprintf(stderr, "사용법: %s 파일이름\n", argv[0]);
12        return 1;
13    }
14
15    fp = fopen(argv[1], "w");
16    printf("%-9s %-7s %-4s\n", "학번", "이름", "점수");
17
18    while (scanf("%d %s %d", &rec.id, rec.name, &rec.score)==3)
19        fprintf(fp, "%d %s %d ", rec.id, rec.name, rec.score);
20
21    fclose(fp);
22    return 0;
23 }
```

포맷 입출력 예제 2

- 파일에서 학생 정보 읽어와서 출력
  - (L15) 인수로 받은 파일을 읽기전용으로 열기
  - **(L18) 파일에 저장된 학생 레코드를 fscanf 함수를 이용해서 읽음**
  - (L19) 읽은 정보를 그대로 printf로 출력

student.h

```
1 struct student {
2     int id;
3     char name[20];
4     short score;
5 };
```

fscan.c

```
1  #include <stdio.h>
2  #include "student.h"
3
4  /* 텍스트 파일에서 학생 정보를 읽어 프린트한다. */
5  int main(int argc, char* argv[])
6  {
7     struct student rec;
8     FILE *fp;
9
10    if (argc != 2) {
11       fprintf(stderr, "사용법: %s 파일이름\n", argv[0]);
12       return 1;
13    }
14
15    fp = fopen(argv[1], "r");
16    printf("%-9s %-7s %-4s\n", "학번", "이름", "점수");
17
18    while (fscanf(fp,"%d %s %d", &rec.id, rec.name, &rec.score)==3)
19       printf("%10d %6s %6d\n", rec.id, rec.name, rec.score);
20
21    fclose(fp);
22    return 0;
23 }
```

4장 강의 목차

- 파일 및 파일 포인터
- 텍스트 파일
- 이진 파일
- 임의 접근
- 버퍼 입출력
- 기타 함수

fopen()에서의 mode

- 이진함수 읽기 관련 mode

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

- 기본 아이디어
  - 어떤 자료형의 데이터(구조체 포함)이던지 그 데이터를 연속된 바이트로 해석해서 파일에서 읽고 쓰는 방식
  - 파일에 저장된 데이터를 연속된 바이트 형태로 읽음 → 원래 변수에 순서대로 저장

```
int fread(void *buf, int size, int n, FILE *fp);
```

fp가 가리키는 파일에서 size 크기의 블록 n개를 읽어서 buf가 가리키는 곳에 저장함
읽어온 블록의 개수를 정수형으로 리턴

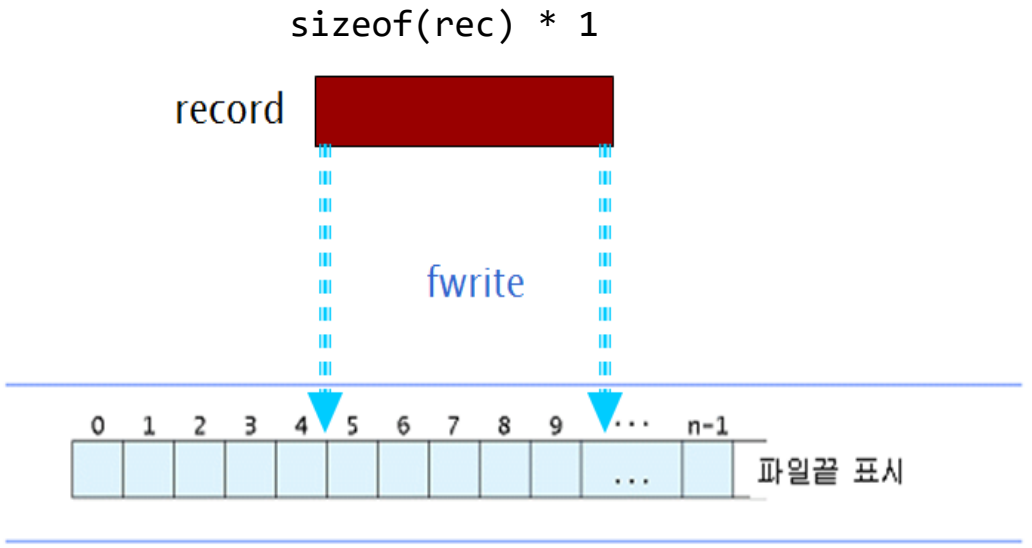
```
int fwrite(const void *buf, int size, int n, FILE *fp);
```

fp가 가리키는 파일에 버퍼 buf에 저장되어 있는 size 크기의 블록 n개를 기록함
성공적으로 기록한 블록 개수를 리턴

블록 단위 파일저장 예제 (1/2)

```c
#include <stdio.h>
#include "student.h"

/* 구조체를 이용하여 학생 레코드를 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);
   }

   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);
}
```

이진 파일 형태로 블록단위로 저장하는 예제



블록 단위 파일저장 예제 (2/2)

```c
#include <stdio.h>
#include "student.h"

/* 구조체를 이용하여 학생 레코드를 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);
   }

   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);
}
```

이진 파일 형태로 블록단위로 저장하는 예제

```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;
}
```

텍스트 파일 형태로 저장하는 예제

블록 단위 저장 및 로드 실습 (1/2)

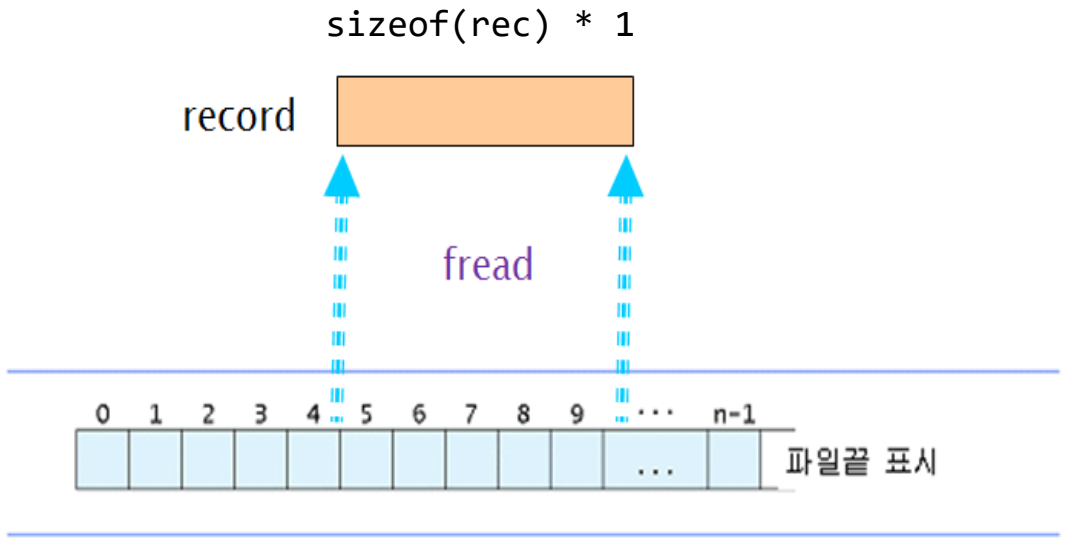
- Check `stcreate1.c` to understand how to use it
  - **`$vim stcreate1.c`**
  - **`$gcc -o stcreate1 stcreate1.c`**
- Execute `stcreate1` to make `student_list.txt` by inputting information about three students
  - You should check **what is defined format** in `scanf`
  - `WARNING:` **Carefully type** the text below. Backspace, arrows would not work.
    - `Student info: 2025001 Lumi 95, 2025002 Joy 90, 2025003 Mira 85`
    - `When you done, do [Ctrl+D]`
- Check the file **`$cat student_list.txt`** (**note: it is a binary file**)

블록 단위 저장 및 로드 실습 (2/2)

- **`$vim stprint.c`** and complete the code, compile, and execute it
  - the code should print out three student's information
- Refer the page #25
  - You should open the file in a "binary mode"
  - You should read row of the file using **`fread`**
  - **`printf`** will be the same

블록 단위 파일로드 예제

```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;
   }

   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;
}
```



이진 파일 형태로 블록단위로 로드하는 예제

블록 단위 파일로드 예제

```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;
   }

   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;
}
```

이진 파일 형태로 블록단위로 로드하는 예제

```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;
}
```

텍스트 파일 형태로 로드하는 예제

4장 강의 목차

- 파일 및 파일 포인터
- 텍스트 파일
- 이진 파일
- 임의 접근
- 기타 함수

임의 접근

- 지금까지의 입출력 함수
  - 파일의 시작부터 끝까지 단방향으로 진행하면서 데이터를 읽거나 씀
  - 파일의 크기가 크거나 원하는 데이터가 파일 여러 군데 흩어져 있을때 비효율적
- 임의 접근
  - 파일 내 원하는 자료가 있는 임의의 위치로 접근하여 입출력을 수행하는 방법
  - 현재 파일 위치 (current file position)
    - 현재 열린 파일에서의 특정 위치로 해당 위치로부터 읽거나 쓰게 됨
  - 파일 위치 포인터 (file position pointer)
    - 시스템 내에서 현재 파일 위치를 가리키는 포인터

파일 위치 관련 함수

fseek(FILE *fp, long offset, int mode);

현재 파일 위치를 기준점 mode(파일 시작, 현재위치, 파일 끝)에서 offset 만큼 이동

rewind(FILE *fp);

fp가 가리키는 파일의 현재 파일 위치를 파일 시작점으로 이동

ftell(FILE *fp);

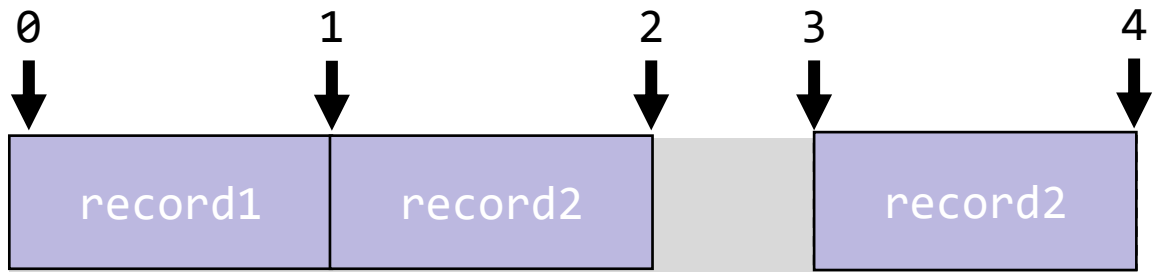
fp가 가리키는 파일의 현재 파일 위치를 반환

fseek() 함수 및 효과 예

- 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

fseek() 함수 및 효과 예

```
0
1 fwrite(&record1, sizeof(record), 1, fp);
2 fwrite(&record2, sizeof(record), 1, fp);
3 fseek(fp, sizeof(record), SEEK_END);
4 fwrite(&record3, sizeof(record), 1, fp);
```



임의 접근을 이용한 학생 레코드 저장: stcreate2.c

- 학번이라는 시스템처럼 1001001부터 시작해서 순차적으로 빠짐없이 정리된다는 가정
  - 1001003은 1001001보다 두 번째 뒷 학번이 됨
  - 포인터가 sizeof(rec)로 건너뛰기 때문에 입력하지 않은 학번은 비어있게 됨
- 코드 간단 설명
  - (L18) scanf로 입력 정보를 받아서
  - (L19) 현재 파일 포인터를 시작 학번 (입력학번-1001001) 만큼 이동시킴
  - (L20) 이동된 파일 포인터에 저장

```
1  #include <stdio.h>
2  #include "student.h"
3  #define START_ID 1001001
4
5  /* 구조체를 이용하여 학생 정보를 파일에 저장한다. */
6  int main(int argc, char* argv[])
7  {
8     struct student rec;
9     FILE *fp;
10    if (argc != 2) {
11       fprintf(stderr, "사용법: %s 파일이름\n", argv[0]);
12       return 1;
13    }
14
15    fp = fopen(argv[1], "wb");
16
17    printf("%7s %6s %4s\n", "학번", "이름", "점수");
18    while (scanf("%d %s %d", &rec.id,    rec.name, &rec.score) == 3) {
19       fseek(fp,  (rec.id – START_ID)* sizeof(rec),  SEEK_SET);
20       fwrite(&rec, sizeof(rec), 1, fp);
21    }
22
23    fclose(fp);
24    return 0;
25 }
```

임의 접근을 이용한 학생 레코드 검색 stquery.c

- stquery.c
  - get a student ID input and print out the related student's information

- Complete the code
  - move file pointer to read the target data using
    - student id,
    - START_ID,
    - fseek()

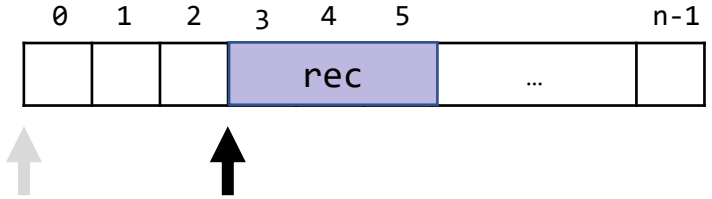
```c
#include <stdio.h>
#include "student.h"
/* Search and print specific student information stored in a file */

int main(int argc, char *argv[])
{
        struct student rec;
        char c;
        int id;
        FILE *fp;

        if (argc != 2) {
                fprintf(stderr, "Usage: %s filename\n", argv[0]);
                return 0;
        }

        if ((fp = fopen(argv[1], "rb")) == NULL) {
                fprintf(stderr, "File open error\n");
                return 2;
        }

        printf("Enter student ID to search: ");
        if (scanf("%d", &id) == 1) {
                // [DO] move file pointer to the related data.
                //(hint: fseek(), refer page 40 in the material)
                if ((fread(&rec, sizeof(rec), 1, fp) > 0) && (rec.id != 0)) {
                        printf("ID: %8d  Name: %4s  Score: %4d\n",
                                        rec.id, rec.name, rec.score);
                } else {
                        printf("Record %d not found\n", id);
                }
        } else {
                printf("Input error\n");
        }

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

임의 접근을 이용한 학생 레코드 수정 `stupdate.c`

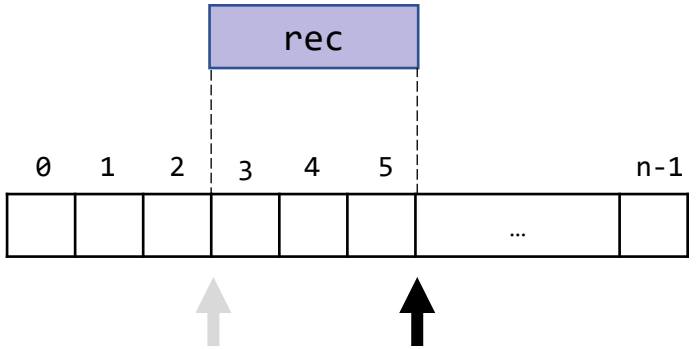
- **[중요]** 기존 데이터를 수정하는 작업은 세 가지 태스크를 가짐
  1. 파일로부터 해당 레코드에 포인터로 접근해서 읽는다
  2. 읽은 정보를 원하는 정보로 수정한다
  3. 수정된 정보를 원래 파일이 있던 포인터로 접근해서 쓴다

- `fread, fwrite, fseek`에 대한 파일 포인터의 흐름을 이해해야 함

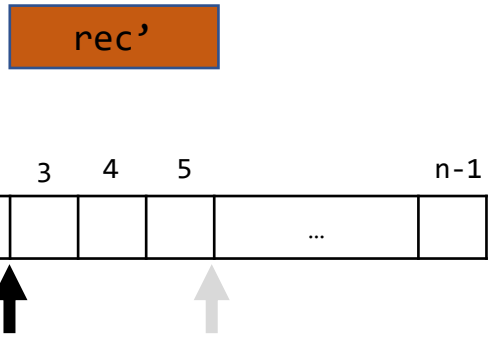
임의 접근을 이용한 학생 레코드 수정 stupdate.c



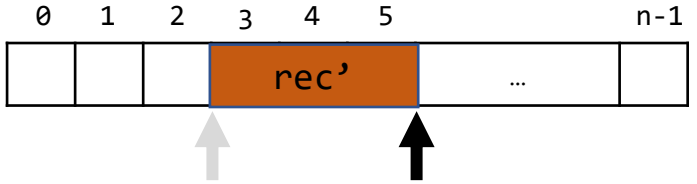
해당 레코드에 접근 (fseek)



rec 구조체 크기만큼 읽음 (fread)
포인터는 fread에 의해 이동됨



rec 정보 수정
같은 위치에 쓰기 위한 파일포인터 이동 (fseek)



수정된 rec’ 쓰기
포인터는 fwrite에 의해 이동됨

임의 접근을 이용한 학생 레코드 수정 stupdate.c

- 주의 사항
  - 임의 접근으로 수정을 원하는 파일로 접근해야 함
  - 파일을 읽고, 수정해서, 원래자리로 써야 함
- 코드 간단 설명
  - (L24-25) 학생 학번을 입력받아서 fseek를 이용하여 해당 학생 정보 위치로 이동
  - (L26) 포인터가 가리키는 곳에서 정보를 읽어 rec에 저장
  - (L31) 같은 위치에 정보를 쓰기 위하여 현재 위치에서 -sizeof(rec) 이동

```
1  #include <stdio.h>
2  #include "student.h"
3  /* 파일에 저장된 학생 레코드를 수정한다. */
4
5  int main(int argc, char *argv[])
6  {
7     struct student rec;
8     int id;
9     char c;
10    FILE *fp;
11
12    if (argc != 2) {
13       fprintf(stderr, "사용법: %s 파일이름\n", argv[0]);
14       return 1;
15    }
16
17    if ((fp = fopen(argv[1], "rb+")) == NULL ) {
18       fprintf(stderr, "파일 열기 오류\n");
19       return 2;
20    }
21
22    do {
23       printf("수정할 학생의 학번 입력: ");
24       if (scanf("%d", &id) == 1) {
25          fseek(fp, (id - START_ID) * sizeof(rec), SEEK_SET);
26          if ((fread(&rec, sizeof(rec), 1, fp) > 0)&&(rec.id != 0)) {
27             printf("학번: %8d 이름: %4s 점수: %4d\n",
28                    rec.id, rec.name, rec.score);
29             printf("새로운 점수 입력: ");
30             scanf("%d", &rec.score);
31             fseek(fp, -sizeof(rec), SEEK_CUR);
32             fwrite(&rec, sizeof(rec), 1, fp);
33          }
34          else printf("레코드 %d 없음\n", id);
35       }
36       else printf("입력오류\n");
37
38       printf("계속하겠습니까?(Y/N)");
39       scanf(" %c",&c);
40    } while (c == 'Y');
41    fclose(fp);
42    return 0;
43 }
```

4장 강의 목차

- 파일 및 파일 포인터
- 텍스트 파일
- 이진 파일
- 임의 접근
- 기타 함수

문자열 처리 함수 (1/2)

- #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 (2/2)

- `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.

마무리

- 핵심 개념 정리
  - 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. 문자단위 입출력: `fgetc()`와 `fputc()`, 한 줄씩 입출력: `fgets()`, `fputs`
  - `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.
  - C 표준 입출력 라이브러리에서는 입출력 최적화를 위해 버퍼 구현이 되어 있다.
- 다음 시간
  - 지금까지는 C라이브러리로 간접적으로 리눅스를 활용하였음
  - 이러한 파일 입출력 내용을 시스템 호출 버전으로 학습함