학번 : _____ 성명 : _____.

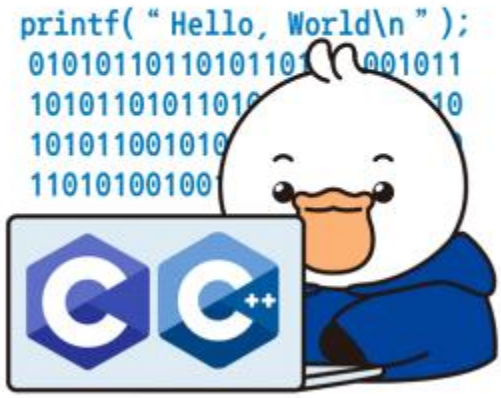
- 본 강의자료는 과학기술정보통신부 및 정보통신기획평가원에서 지원하는『소프트웨어중심대학』사업의 결과물입니다.
- 본 강의자료는 내용은 전재할 수 없으며, 인용할 때에는 반드시 과학기술정보통신부와 정보통신기획평가원의 '소프트웨어중심대학'의 결과물이라는 출처를 밝혀야 합니다.

SW 대구가톨릭대학교 소프트웨어중심대학사업단

대구가톨릭대학교 컴퓨터소프트웨어학부 School of Computer Software, Daegu Catholic University

Part 13. 파일처리

목차



13.1 파일 스트림

- 파일 [ ]
  - ✓ [ ] 저장매체에 저장된 데이터
  - ✓ 기기에서 [ ] 정보를 담는 논리적인 단위
  - ✓ 휘발성 메모리 - [ ]
    - ➢ 프로그램 종료 시 데이터가 사라짐
  - ✓ 비휘발성 메모리 – [ ]
    - ➢ 프로그램 종료 시 데이터가 보존됨

13.1 파일 스트림

- 파일의 종류
  - ✓ 텍스트 파일[ ]
    - ➢ 메모장과 같은 텍스트 편집기를 사용하여 작성된 파일
    - ➢ 문자 코드 값으로 구성되어 저장
      - 문자 코드 값 : [ ], [ ] 등
    - ➢ [ ]도 문자 형식으로 변환되어 저장되는 특징

13.1 파일 스트림

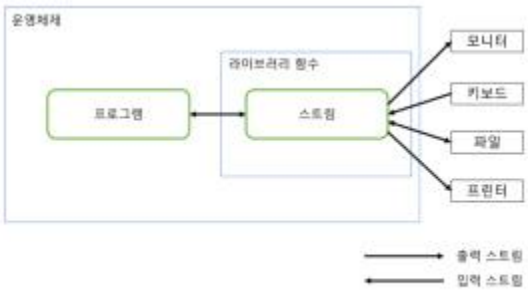
- 파일의 종류
  - ✓ 이진 파일[ ]
    - ➢ 프로그램 실행 파일, 사진 또는 그림 파일(jpg, jpeg, gif 등), 음악 파일(mp3, wav 등), 동영상 파일(mp4, avi, mov 등), 문서 파일(hwp, pptx, xlsx 등)
    - ➢ 각 프로그램의 목적에 따라 다른 방식으로 [ ]의 자료를 저장
      - 한글에서 작성한 파일 - hwp 형식으로 저장
      - 파워포인트에서 작성한 파일 - pptx 형식으로 저장

13.1 파일 스트림

- 파일의 종류
  - ✓ 파일에 저장되는 데이터의 구성과 형식은 각 프로그램에 따라 상이
    - ➢ 텍스트 파일
      - 문자를 표현하기 위한 [ ] 방식 사용
      - 데이터 [ ] 과정 필요
      - 운영체제의 [ ] 에서 파일 내용 확인 및 수정 가능
    - ➢ 바이너리 파일
      - 프로그램에서 사용하는 데이터를 [ ] 과정 없이 저장
      - 빠르게 [ ] 가능
      - 별도의 [ ] 사용하야 파일 내용 확인 및 수정 가능

13.1 파일 스트림

- 입출력 스트림
  - ✓ 스트림 [ ]
    - ➢ 실제 입출력 데이터의 [ ] 역할을 하는 별도의 장치 또는 프로그램
    - ➢ C 언어에서는 파일이나 콘솔의 입출력을 [ ] 을 통해 제어



13.1 파일 스트림

- 입출력 스트림
  - ✓ 출력 스트림
    - ➢ 프로그램에서 [ ]을 통해 외부 장치나 파일, 네트워크로 나가는 경로
  - ✓ 입력 스트림
    - ➢ 외부 장치나 파일, [ ]부터 들어오는 경로
  - ✓ 스트림은 [ ]와 밀접하게 관련, [ ]에서 직접 관리
  - ✓ 읽기/쓰기 권한(Read/Write Access) 설정
    - ➢ [ ]로부터 입력과 출력에 대한 권한 여부 정보
      - [ ]: 키보드와 마우스, 모니터, 프린터 등

13.1 파일 스트림

- 입출력 스트림
  - ✓ 버퍼[ ]
    - ➢ 메모리의 일부분, 장치나 파일의 입출력에 데이터를 [ ]하는 곳
    - ➢ unbuffered
      - 어떠한 상태에서도 데이터의 입출력이 [ ]으로 이루어지는 스트림
      - 중간 저장 공간인 [ ] 사용하지 않음
    - ➢ fully buffered
      - 입출력 데이터가 [ ]에 가득 차야 데이터의 입출력이 진행되는 스트림
    - ➢ line buffered
      - [ ]가 버퍼에 들어왔을 때, 입출력이 진행되는 스트림
    - ➢ 표준 출력[ ]: line buffered 또는 Fully buffered 스크림 사용
    - ➢ 표준 출력[ ]: fully buffered 스크림 사용
    - ➢ 표준 에러[ ]: unbuffered 스크림 사용

13.1 파일 스트림

- 입출력 스트림
  - ✓ 상태 표시자
    - ➢ 를 표시하기 위한 특정한 표시자
    - ➢ 입출력에 영향을 줄 수 있음
    - ➢ 오류 표시자
      - 스트림과 관련된 에서 오류 발생 시 설정됨
      - `ferror()` 함수 : 표시자의 상태 확인
      - `clearerr()`, `rewind()`, `fseek()`, `fsetpos()` 함수 : 표시자의 상태
    - ➢ 파일 끝 표시자
      - 스트림의 입출력 연산이 파일의 끝 부분에 을 의미
      - `feof()` 함수 : 여부 확인
      - `clearerr()`, `rewind()`, `fseek()`, `fsetpos()` 함수 : 표시자의 상태

13.1 파일 스트림

- 입출력 스트림
  - ✓ 상태 표시자
    - ➢ 위치 표시자
      - 포인터를 이용해 스트림 입출력 연산에서 또는 를 가리킴
      - `ftell()`, `fgetpos()` 함수 : 가 가리키는 값 확인
      - `clearerr()`, `rewind()`, `fseek()`, `fsetpos()` 함수 : 표시자의 상태

13.1 파일 스트림

- 파일 스트림
  - ✓ 프로그램에서 [ ]에 저장된 파일의 정보를 [ ]하기 위해 파일 스트림을 연결
  - ✓ stdio.h
    - ➢ 파일 입출력을 위한 [ ] 제어 라이브러리 함수가 정의되어 있음

13.1 파일 스트림

- 파일 스트림
  - ✓ stdio.h 헤더파일의 스트림 제어 라이브러리 함수 종류

	바이트 문자	확장 문자	설명
파일 접근	fopen		파일 열기(윈도우에선 비-유니코드 파일 이름, 유닉에서는 UTF-8 파일 이름)
	freopen		존재하는 스트림으로 다른 파일 열기
	fflush		대응되는 파일로 출력 스트림 동기화하기
	fclose		파일 닫기
	setbuf		파일 스트림에 버퍼 장착시키기
	setvbuf		파일 스트림 크기에 맞게 버퍼 장착
	fwide		전각 문자와 반각 문자간 파일 스트림 교환
직접 입출력	fread		파일 읽기
	fwrite		파일 쓰기
언포맷 입출력	fgetc getc	fgetwc getwc	파일 스트림으로 부터 바이트/wchar_t 읽기
	fgets	fgetws	파일 스트림으로부터 바이트/wchar_t 라인 읽기
	fputc	fputwc	파일 스트림에 바이트/wchar_t 쓰기
	fputs	fputws	파일 스트림에 바이트/wchar_t 문자열 입력
	ungetc	ungetwc	파일 스트림에 바이트/wchar_t 제자리에 돌려놓기

13.1 파일 스트림

- 파일 스트림
  - ✓ stdio.h 헤더파일의 스트림 제어 라이브러리 함수 종류

	바이트 문자	확장 문자	설명
포맷 입출력	scanf fscanf sscanf	wscanf fwscanf swscanf	파일 스트림이나 버퍼의 표준 입력으로부터 형식화된 바이트/wchar_t 입력읽기
	vscanf vfscanf vsscanf	vwscanf vfwscanf vswscanf	가변 인자 목록을 쓰는 파일 스트림이나 버퍼의 표준 입력으로부터 형식화된 바이트/wchar_t 읽기
	printf fprintf sprintf snprintf	wprintf fwprintf swprintf	파일 스트림이나 버퍼의 표준 출력으로 형식화된 바이트/wchar_t 출력을 출력하기
	vprintf vfprintf vsprintf vsnprintf	vwprintf vfwprintf vswprintf	가변 인자 목록을 쓰는 파일 스트림이나 버퍼의 표준 출력으로 형식화된 바이트/wchar_t 출력을 출력
	perror		표준 에러의 현재 에러 설명 쓰기

13.1 파일 스트림

- 파일 스트림
  - ✓ stdio.h 헤더파일의 스트림 제어 라이브러리 함수 종류

	바이트 문자	확장 문자	설명
파일 위치 조정	ftell / ftello		현재 파일 포인터 되돌려주기
	fseek fseeko		파일의 특정 위치로 파일 포인터 이동
	fgetpos		파일 포인터 얻기
	fsetpos		파일의 특정 위치로 파일 포인터 이동
	rewind		파일 포인터를 파일의 첫 시작 부분으로 이동

13.1 파일 스트림

- 파일 스트림
  - ✓ stdio.h 헤더파일의 스트림 제어 라이브러리 함수 종류

	바이트 문자	확장 문자	설명
에러 조작	clearerr		에러 삭제
	feof		파일의 끝 체크
	ferror		파일 에러 체크
파일 조작 명령	remove		파일 삭제
	rename		파일 이름 수정
	tmpfile		임시 파일로 포인터 되돌리기
	tmpnam		특수 파일 이름 되돌려주기

13.1 파일 스트림

- 파일 스트림
  - ✓ [ ]
    - ➢ 보조기억장치에 저장된 파일에 접근하여 파일 스트림을 연결하기 위한 함수
    - ➢ filname 인자 값으로 들어온 파일을 mode 인자의 값으로 모드로 설정
    - ➢ 파일 스트림을 열어 FILE 객체를 가리키는 포인터 반환
    - ➢ mode 인자에 의해 해당 스크림에서 어떠한 입출력 작업이 가능한지 결정됨
    - ➢ 함수 원형

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

13.1 파일 스트림

- 파일 스트림
  - ✓ [ ]를 제어하기 위한 모드
  - ✓ mode 인자 종류 및 설명

mode	설명
"r"	읽기 형식으로 연다. (파일은 반드시 존재해야함)
"w"	쓰기 형식으로 연다. (동일 파일 존재 시, 그 파일의 내용을 모두 지우고 새로운 빈 파일로 간주, 파일이 없다면 빈파일 새롭게 생성)
"a"	파일을 덧붙이기(append)형식으로 연다.(쓰기 작업은 파일 끝에 데이터를 덧붙임, 파일이 없다면 빈파일 새롭게 생성)
"r+"	읽기 형식으로 연다. (파일은 반드시 존재해야함) / (모드 변환 가능)
"w+"	쓰기 형식으로 연다. (동일 파일 존재 시, 그 파일의 내용을 모두 지우고 새로운 빈 파일로 간주, 파일이 없다면 빈파일 새롭게 생성) / (모드 변환 가능)
"a+"	파일을 덧붙이기(append)형식으로 연다.(쓰기 작업은 파일 끝에 데이터를 덧붙임, 파일이 없다면 빈파일 새롭게 생성) / (모드 변환 가능)

13.1 파일 스트림

소스 13-1

```
1  #include <stdio.h>
2  int main(void) {
3     FILE* file_p,* file_p2;
4     if ((file_p = fopen("test.txt", "w")) == NULL) {
5        printf("오류!! 파일을 읽어오지 못했습니다.\n");
6        return 0;
7     }
8     fclose(file_p);
9     return 0;
10 }
```

출력 예

```
오류!! 파일을 읽어오지 못했습니다.
```

13.1 파일 스트림

- 파일 스트림

✓ [ ]

- 파일 스크림을 닫는 함수
- _File 매개변수에 현재 열려있는 파일 스트림을 인자로 전달하여 파일 스트림 종료
- 정상적으로 호출되었을 경우
  - 파일 스트림 연결에 할당된 자원을 반납하고 버퍼의 내용을 모두 지움
- 정상적으로 종료되었을 경우
  - 0 반환, 실패할 경우 EOF 반환
- 함수 원형

```
#include <stdio.h>
int [      ](FILE* _File);
```


13.1 파일 스트림

- 멤버 상수

✓ 파일 입출력 제어를 위한 stdio.h의 멤버 상수

이름	설명
EOF	end-of-file 가리키는 용도로 사용되는 int 형의 음의 정수
BUFSIZ	setbuf() 함수에 의해 버퍼 크기를 나타내는 정수
FILENAME_MAX	충분히 열 수 있는 저장 가능한 파일 이름의 char 형의 배열 크기
FOPEN_MAX	동시에 열 수 있는 파일의 개수; 최소 8
_IOFBF	"input/output fully buffered"의 약어. 이 정수값은 setvbuf() 함수에 넘겨 버퍼화된 블록의 스트림 요청
_IOLBF	"input/output line buffered"의 약어. 이 정수값은 setvbuf() 함수에 넘겨 버퍼화된 블록의 스트림 요청
_IONBF	"input/output unbuffered"의 약어. 이 정수값은 setvbuf() 함수에 넘겨 버퍼화된 블록의 스트림 요청
L_tmpnam	char 배열이 tmpnam() 함수를 발생시킬 정도의 충분한 크기
NULL	널 포인터의 약어인 매크로 상수. 이 상수는 메모리의 어떤 유효한 위치의 개체도 가리키지 않는 포인터 값
SEEK_CUR	현재 파일 위치에 대해 위치 변경을 요청하는 fseek()에 전달되는 정수
SEEK_END	파일의 끝에 대해 위치 조정을 요청하기 위한 fseek() 함수에 전달되는 정수.
SEEK_SET	파일의 시작 위치를 기준으로 한 위치 지정을 요청하기 위한 fseek() 함수에 전달되는 정수
TMP_MAX	tmpnam() 기능에 의해 만들어지는 특수 파일이름의 최대길이 (최소 25자)

13.2 텍스트 파일 입출력

- 텍스트 정보를 저장하기 위해 사용되는 라이브러리 함수
  - ✓ stdio.h 헤더파일의 텍스트 정보 저장 라이브러리 함수 종류 및 설명

	바이트 문자	확장 문자	설명
언포맷 입출력	fgetc	fgetwc	파일 스트림으로부터 바이트/wchar_t 읽기
	fgets	fgetws	파일 스트림으로부터 바이트/wchar_t 라인 읽기
	fputc	fputwc	파일 스트림에 바이트/wchar_t 쓰기
	fputs	fputws	파일 스트림에 바이트/wchar_t 문자열 입력
	ungetc	ungetwc	파일 스트림에 바이트/wchar_t 제자리에 돌려놓기

13.2 텍스트 파일 입출력

- 텍스트 정보를 저장하기 위해 사용되는 라이브러리 함수
  - ✓ stdio.h 헤더파일의 텍스트 정보 저장 라이브러리 함수 종류 및 설명

	바이트 문자	확장 문자	설명
포맷 입출력	scanf fscanf sscanf	wscanf fwscanf swscanf	파일 스트림이나 버퍼의 표준 입력으로부터 형식화된 바이트/wchar_t 입력읽기
	vscanf vfscanf vsscanf	vwscanf vfwscanf vswscanf	가변 인자 목록을 쓰는 파일 스트림이나 버퍼의 표준 입력으로부터 형식화된 바이트/wchar_t 읽기
	printf fprintf sprintf snprintf	wprintf fwprintf swprintf	파일 스트림이나 버퍼의 표준 출력으로 형식화된 바이트/wchar_t 출력을 출력하기
	vprintf vfprintf vsprintf vsnprintf	vwprintf vfwprintf vswprintf	가변 인자 목록을 쓰는 파일 스트림이나 버퍼의 표준 출력으로 형식화된 바이트/wchar_t 출력
	perror		표준 에러의 현재 에러 설명 쓰기

13.2 텍스트 파일 입출력

- 데이터를 파일에 입출력하는 방식
  - ✓ 언포맷 입출력
    - ➢ 입출력 수행 시 형식을 [ ] 하지 않고 사용하는 방식
  - ✓ 포맷 입출력
    - ➢ 명시적으로 입출력 포맷에 대하여 [ ] 하여 사용하는 방식
  - ✓ 앞 표의 라이브러리 함수
    - ➢ 프로그램의 입출력 동작 수행 시
      - 컴퓨터에게 입력 또는 출력에 대해 [ ] 으로 선언 과정 필요

13.2 텍스트 파일 입출력

- 표준 파일의 종류

표준 파일	키워드	장치
표준 입력	stdin	키보드
표준 출력	stdout	모니터 콘솔 화면
표준 에러	stderr	모니터 콘솔 화면

  - ✓ stdin
    - ➢ [ ] 의 약어, 키보드를 통한 표준 입력 수행할 때 사용
  - ✓ stdout
    - ➢ [ ] 의 약어, 모니터 콘솔을 통해 출력할 때 사용
  - ✓ stderr
    - ➢ [ ] 의 약어, 표준 입출력 간 발생한 에러 표기를 위해 사용

13.2 텍스트 파일 입출력

- [ ] 입출력 함수
  - ✓ 입출력을 위한 대표적인 [ ] 입출력 함수
    - ➢ fgetc() : 표준 입출력 또는 파일 내 정보 읽기
    - ➢ fputc() : 정보 출력
      - 파일 스트림에서 입력된 문자 [ ]개를 입력받아 정수형 타입 int로 [ ]를 반환
      - 읽는 도중 에러 발생 시, [ ]반환
    - ➢ fputs() : 정보 출력

```c
#include <stdio.h>
int [ ](FILE* stream);
int [ ](int character, FILE* stream);
int [ ](const char* str, FILE* stream);
```


13.2 텍스트 파일 입출력

소스 13-2

```c
#include <stdio.h>
int main(void) {
  int i = fgetc(stdin);
  printf("표준 입력 결과 : %c \n", i);
  return 0;
}
```

입력 예	a
출력 예	표준 입력 결과 : a

13.2 텍스트 파일 입출력

- [ ] 입출력 함수
  - ✓ fputc()
    - ➢ 스트림에서 [ ]를 읽어 출력하는 함수
    - ➢ 매개변수
      - int character : 파일 스트림에서 작성할 문제를 인자로 받음
      - File* stream : 문자가 작성될 스트림 자체를 인자로 받음
  - ✓ fputs()
    - ➢ 문자열을 전달받아 [ ]에 전달

13.2 텍스트 파일 입출력

소스 13-3

```
1  #include <stdio.h>
2  int main(void) {
3     char str[20] ="C Programing\n";
4     int index =0;
5     while (str[index]) {
6        fputc(str[index], stdout);
7        index++;
8     }
9     fputs(str, stdout);
10    return 0;
11 }
```

출력 예

```
C Programing
C Programing
```

13.2 텍스트 파일 입출력

소스 13-4

```c
#include <stdio.h>
int main(void) {
   FILE* file_p;
   if ((file_p = fopen("test.txt", "w")) == NULL) {
      printf("오류!! 파일을 읽어오지 못했습니다.\n");
      return 0;
   }
   fputs("Hi my name is yunsung.\nHello World\n",file_p);
   fclose(file_p);
   return 0;
}
```

출력 예 (test.txt)	Hi my name is yunsung. Hello World

13.2 텍스트 파일 입출력

- [ ]입출력 함수
  - ✓ [ ], [ ]
    - ➢ printf(), scanf() 와 사용법 유사
      - 차이점 : 첫 번째 인자로 파일 스트림이 추가됨
      - File* stream 인자로 stdin 또는 stdout 전달 시, printf(),scanf() 함수와 동일하게 작동
    - ➢ 매개변수
      - FILE* stream : 입출력에 사용될 파일 스트림
      - const char* format : 입출력을 제어하기 위한 C 형식의 문자열
      - 가변인자 : 여러 개의 출력 또는 입력을 수행할 변수 또는 상수

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

13.2 텍스트 파일 입출력

소스 13-5

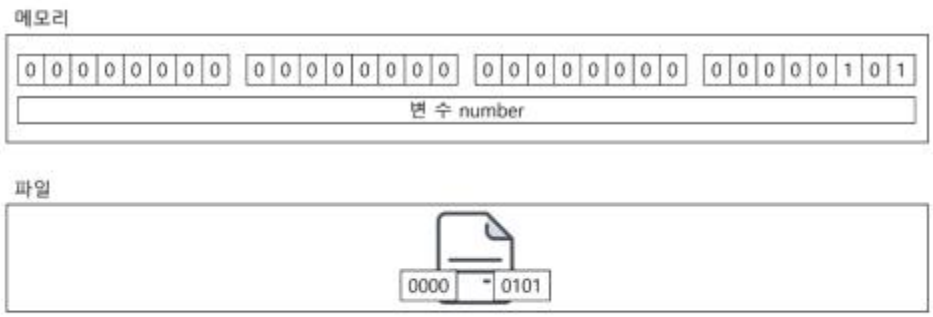
```c
#include <stdio.h>
int main(void) {
    char str[80];
    int input_integer;
    FILE* pFile;
    pFile = fopen("fileIO.txt", "w+");
    fprintf(pFile, "%d %s", 123456, "file_io_test");
    fclose(pFile);
    pFile = fopen("fileIO.txt", "r");
    fscanf(pFile, "%d", &input_integer);
    fscanf(pFile, "%s", str);
    fclose(pFile);
    printf("I have read: %d and %s \n", input_integer, str);
    return 0;
}
```

출력 예 (stdout)	I have read: 123456 and file_io_test
출력 예 (FileIO.txt)	123456 file_io_test

13.3 바이너리 파일 입출력

- 바이너리 입출력
  - ✓ 메모리상에 [ ]로 저장된 정보가 별도의 작업없이 C 언어의 자료형을 모두 유지하면서 [ ] 단위의 파일을 입력 및 출력을 수행

```c
int number = 5;
fprintf(f, "%d", number)
```



13.3 바이너리 파일 입출력

- 바이너리 파일 입출력을 위한 라이브러리 함수

✓ [ ]

➢ 파일에서부터 [ ] 단위로 저장된 값을 읽어오는 함수

- `void* ptr` : 파일에서 읽은 값을 저장할 변수의 주소
- `size_t size` : 읽은 값의 자료의 크기
- `size_t count` : 출력될 항목의 개수
- `FILE* stream` : 출력할 파일 포인터

✓ [ ]

➢ 파일에 자료를 저장하기 위해 사용되는 함수

- 매개변수 역할 [ ] 와 동일

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

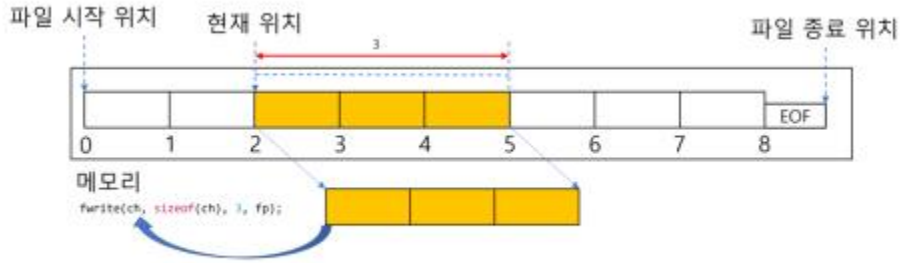

13.3 바이너리 파일 입출력

- 바이너리 파일 입출력을 위한 라이브러리 함수

✓ [ ], [ ]

➢ 파일을 [ ] 단위로 저장된 값 읽음

- 현재 파일을 읽는 [ ] 부터, 두 번째 인자값으로 지정한 [ ] 만큼, 세 번째 인자값으로 지정한 [ ] 를 들고 오는 것



13.3 바이너리 파일 입출력

- 바이너리 파일 제어 모드의 종류 및 설명
  - ✓ 이진(binary) 값 입출력을 수행하는 모드로 □ 추가

mode	설명
"rb"	이진파일 읽기 형식으로 연다. (파일은 반드시 존재해야함)
"wb"	이진파일 쓰기 형식으로 연다. (동일 파일 존재 시, 그 파일의 내용을 모두 지우고 새로운 빈 파일로 간주, 파일이 없다면 빈 파일 새롭게 생성)
"ab"	이진파일을 덧붙이기(append)형식으로 연다.(쓰기 작업은 파일 끝에 데이터를 덧붙임, 파일이 없다면 빈 파일 새롭게 생성)

13.3 바이너리 파일 입출력

소스 13-6

```c
#include <stdio.h>
#include <string.h>
struct BinaryFileIO {
    char str[32];
    int number;
};
int main(void) {
    FILE* fp;
    struct BinaryFileIO f1,f2,f3;
    strcpy(f1.str, "test");
    strcpy(f2.str, "test");
    strcpy(f3.str, "test");
    f1.number =1;
    f2.number =2;
    f3.number =3;
    fp = fopen("file.txt", "w+b");
    if (fp !=NULL) {
        fwrite(&f1, 1, sizeof(BinaryFileIO), fp);
        fwrite(&f2, 1, sizeof(BinaryFileIO), fp);
        fwrite(&f3, 1, sizeof(BinaryFileIO), fp);
        fclose(fp);
    }
}
```

출력 예 (file.txt)

```
test 烤烤烤烤烤烤烤烤烤烤烤烤?  test
烤烤烤烤烤烤烤烤烤烤烤烤?  test
烤烤烤烤烤烤烤烤烤烤烤烤?
```

13.3 바이너리 파일 입출력

- 바이너리 형태로 저장되어 있는 [ ]
  - ✓ [ ] 되어 있지 않음
  - ✓ [ ] 에서 확인 불가
  - ✓ [ ] 의 값 확인하기
    - ➢ hexa 에디터
      - [ ] 진수로 편집 가능
      - fread() 함수를 이용해 값 읽을 때 문제 없음

13.3 바이너리 파일 입출력

소스 13-1

```
#include <stdio.h>
#include <string.h>
struct BinaryFileIO {
    char str[32];
    int number;
};
int main(void) {
    FILE* fp;
    struct BinaryFileIO file;
    int len;
    fp = fopen("file.txt", "r");
    if (fp !=NULL) {
        while (!feof(fp))
        {
            len = fread(&file, sizeof(BinaryFileIO), 1, fp);
            if (ferror(fp) || !len)break;
                printf("read : %s, %d\n", file.str, file.number);
        }
        fclose(fp);
    }
}
```

출력 예	read : test, 1 read : test, 2 read : test, 3

13.4 고급 파일 입출력

- 파일을 제어하는 관련 라이브러리 함수
  - ✓ 실제 프로그램에서 파일의 처음부터 [ ]까지 읽는 형태의 입출력보다는 [ ] 부분을 읽고 쓰는 형태의 동작이 빈번하게 일어남
  - ✓ [ ]
    - ➢ 파일에 대한 접근과 에러, 내부 위치를 제어

13.4 고급 파일 입출력

- 파일을 제어하는 관련 라이브러리 함수
  - ✓ 파일 입출력과 관련된 고급 보조 함수 종류 및 설명

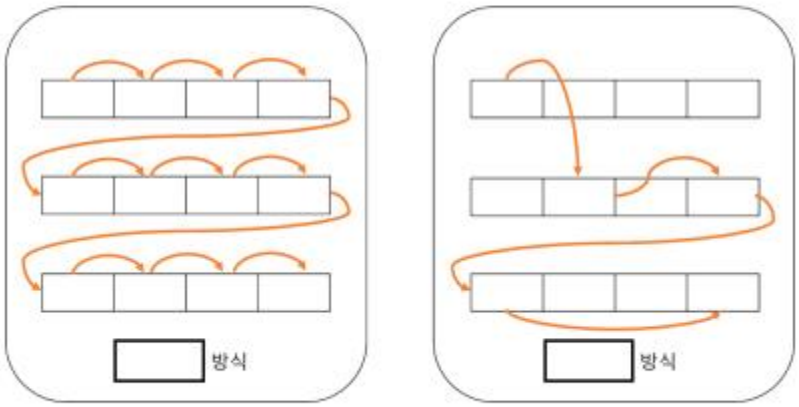
	함수	설명
파일 위치 조정	ftell / ftello	현재 파일 포인터 되돌려주기
	fseek / fseeko	파일의 특정 위치로 파일 포인터 이동
	fgetpos	파일 포인터 얻기
	fsetpos	파일의 특정 위치로 파일 포인터 이동
	rewind	파일 포인터를 파일의 첫 시작 부분으로 이동
에러 조작	clearerr	에러 삭제
	feof	파일의 끝 체크
	ferror	파일 에러 체크
파일 조작 명령	remove	파일 삭제
	rename	파일 이름 수정
	tmpfile	임시 파일로 포인터 되돌리기
	tmpnam	특수 파일 이름 되돌려주기

13.4 고급 파일 입출력

- 파일 위치 조정
  - ✓ 프로그램에서 파일 열 때, 모드에 관계없이 파일 시작점[ ]에서 시작
  - ✓ 순차 접근(sequential access) 방식
    - ➢ [ ]으로 파일의 위치를 참조, [ ] 참조하여 접근하는 방식
  - ✓ 임의 접근(random access) 방식
    - ➢ [ ] 위치를 제어하면서 [ ] 부분만 접근하여 읽고 쓰는 방식

13.4 고급 파일 입출력

- 파일 위치 조정
  - ✓ 순차 접근과 임의 접근



13.4 고급 파일 입출력

- 임의 접근 : fseek() 함수
  - ✓ 파일의 임의 접근을 위해 [ ]위치를 임의로 이동하게 하는 함수

```
#include <stdio.h>
int [ ](FILE* stream, long int offset, int origin);
```

  - ✓ 매개변수
    - ➢ 첫 번째 : 파일 스트림을 인자로 받음
    - ➢ 세 번째 : 기준으로 이동할 위치 정보를 인자로 받음

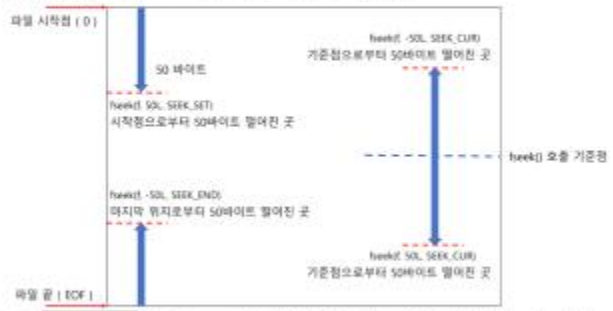
13.4 고급 파일 입출력

- 임의 접근 : fseek() 함수
  - ✓ 파일 위치 설정하는 값 : [ ]에 정의된 매크로 상수 이용
  - ✓ 파일 위치 설정을 위한 매크로 상수 종류

이름	값	해설
SEEK_CUR	0	현재 파일 위치에 대해 위치 변경을 요청하는 fseek()에 전달되는 정수
SEEK_END	1	파일의 끝에 대해 위치 조정을 요청하기 위한 fseek() 함수에 전달되는 정수.
SEEK_SET	2	파일의 시작 위치를 기준으로 한 위치 지정을 요청하기 위한 fseek() 함수에 전달되는 정수.

13.4 고급 파일 입출력

- 임의 접근 : fseek() 함수
  - ✓ 파일 위치 설정을 위한 매크로 상수 종류



  - ✓ 파일 시작점 기준, [ ] 단위로 이동하여 현재 파일 내에서 가리키고 있는 위치를 임의로 이동
  - ✓ 파일 이동 단위 : 1L(long) 당 [ ] 이동

13.4 고급 파일 입출력

- 임의 접근 : ftell() 함수, rewind() 함수
  - ✓ ftell() 함수
    - ➢ 현재 파일 내에서 [ ] 위치 반환
  - ✓ rewind() 함수
    - ➢ 파일의 위치를 [ ] 으로 이동

```
#include <stdio.h>
long int ftell(FILE* stream);
void rewind(FILE* stream);
```

  - ✓ 매개변수
    - ➢ 현재 파일 스트림을 인자로 사용

13.4 고급 파일 입출력

소스 13-8

```c
#include <stdio.h>
int main(void) {
    FILE* fp;
    fp = fopen("file.txt", "w+");
    if (fp ==NULL) puts("파일을 생성할 수 없습니다.");
    else {
        fputs("abcde", fp);
        printf("파일 포인터의 위치 : %d \n", ftell(fp)); // 위치:5
        fseek(fp, -2L, SEEK_CUR);
        printf("파일 포인터의 위치 : %d \n", ftell(fp)); // 위치:3
        rewind(fp);
        printf("파일 포인터의 위치 : %d \n", ftell(fp)); // 위치:0
        fclose(fp);
    }
}
```

출력 예

```
파일 포인터의 위치 : 5
파일 포인터의 위치 : 3
파일 포인터의 위치 : 0
```


13.4 고급 파일 입출력

- 파일 에러 제어
  - ✓ [ ] 과정에서 발생하는 에러 제어를 위한 함수

	함수	설명
에러 조작	clearerr	에러 삭제
	feof	파일의 끝 체크
	ferror	파일 에러 체크

13.4 고급 파일 입출력

- 파일 에러 제어
  - ✓ [ ] 함수
    - ➢ 입출력 과정에서 발생한 오류를 초기화
  - ✓ [ ] 함수
    - ➢ 파일 입력에서 현재 파일 스트림이 읽는 자료가 파일의 끝 표시자인지 확인
    - ➢ 파일 끝에 도달 시, 0 반환
  - ✓ [ ] 함수
    - ➢ 입출력 과정에서 발생한 오류 확인, 오류가 있다면 0 반환

13.4 고급 파일 입출력

- 파일 에러 제어
  - ✓ 함수 원형

```c
#include <stdio.h>
void clearerr(FILE* stream);
int feof(FILE* stream);
int ferror(FILE* stream);
```

    - ➢ 매개변수
      - 현재 연결되어 있는 파일 스트림을 인자로 전달하여 파일 스트림의 입출력 에러를 제어

13.4 고급 파일 입출력

- 파일 조작 명령
  - ✓ [ ]을 통해 생성한 파일이나 기존 파일에 접근해, 파일의 삭제와 파일 이름 수정 등의 작업을 수행하기 위해 사용
  - ✓ 조작 명령어 : [ ]에 정의되어 있음
  - ✓ 파일 조작 제어 함수 종류 및 설명

	함수	설명
파일 조작 명령	remove	파일 삭제
	rename	파일 이름 수정
	tmpfile	임시 파일로 포인터 되돌리기
	tmpnam	특수 파일 이름 되돌려주기

13.4 고급 파일 입출력

- 파일 조작 명령

```c
#include <stdio.h>
int remove(const char *filename);
int rename(const char *old, const char *new);
FILE *tmpfile(void);
char *tmpnam(char *s);
```

  - ✓ [ ] 함수
    - ➢ 인자값으로 전달한 파일 삭제, 더는 접근할 수 없도록 만드는 함수
    - ➢ 정상적으로 삭제 시 0 반환, 실패 시 0이 아닌 값 반환
  - ✓ [ ] 함수
    - ➢ 인자로 받은 현재 파일명과 변경할 파일명을 이용하여 파일명을 변경하는 함수
    - ➢ 성공적으로 이름 변경 시 0 반환, 실패 시 0이 아닌 값 반환

13.4 고급 파일 입출력

- 파일 조작 명령
  - ✓ [ ] 함수
    - ➢ 프로그램이 비정상적으로 종료되는 경우 파일 복구를 위해 임시 바이너리 파일을 생성하는 함수
    - ➢ 성공적으로 파일 생성 시 파일 스트림 반환, 실패 시 null 포인터 반환
  - ✓ [ ] 함수
    - ➢ 파일 생성 시 기존 파일과 파일명이 겹치지 않는 파일을 생성하기 위해 사용되는 함수
    - ➢ 정상적으로 동작 시 무작위 문자열 반환, 실패 시, null 포인터 반환

13.5 Q&A

Q 50. 구조체를 화일에서 읽거나 쓰는 방법은?

A . 구조체를 화일에 쓸 경우는 대개 fwrite() 함수를 사용한다.

```
fwrite(&somestruct, sizeof somestruct, 1, fp);
```

이렇게 쓴 데이터는 fread()를 써서 읽을 수 있다. 그러면 fwrite가 구조체를 가리키고 있는 포인터를 써서, 주어진 구조체가 저장되어있는 메모리의 내용을 파일에 기록한다. 이때 sizeof 연산자는 복사할 byte 수를 지정한다. ANSI 호환의 컴파일러를 쓰고, 함수 선언이 되어 있는 헤더 파일을 (대개 <stdio.h>) 포함했으면 위와 같이 쓰는 것이 좋다. 만약 ANSI 이전의 컴파일러를 쓰고 있다면, 첫 번째 인자를 다음과 같이 캐스팅해주어야 한다.

```
fwrite((char *)&somestruct, sizeof somestruct, 1, fp);
```

여기서 중요한 것은, fwrite가 구조체에 대한 포인터가 아니라, 바이트를 가리키는 포인터를 받는다는 것이다. 위와 같이 쓰여진 데이터 파일은 이식성이 없다. 구조체가 저장되는 메모리 이미지는 컴퓨터와 컴파일러에 매우 의존적이다. 각각 다른 컴파일러는 각각 다른 크기의 padding을 사용할 수 있으며, 바이트 크기와 순서(endian : 엔디안)가 다를 수 있다.

13.5 Q&A

Q 51. 아주 간단한 fopen 조차도 동작하지 않는다. 아래 코드에서 잘못된 점이 있는가?

```
FILE *fp = fopen(filename, 'r');
```

A. 본 코드의 문제는 fopen의 두 번째 인자인 mode는 "r"과 같은 string이어야지, 'r'과 같은 문자가 아니라는 것이다.

13.5 Q&A

Q 52. 왜 절대 경로를 써서 파일을 열면 항상 실패할까? 다음 코드는 동작하지 않는다.

```
fopen("c:\newdir\file.dat", "r");
```

A. 두 개의 백슬래시(backslash) 문자를 써야 한다. 질문 19.17을 참고하기 바랍니다.

13.5 Q&A

Q 53. 사용자에게 입력 파일이 없다는 경고를 출력하고 싶다. 파일이 존재하는지 어떻게 하면 검사할 수 있을까?

A. 이런 간단한 문제도 표준에 맞게, 또는 호환성이 높게 처리할 방법이 없다는 것은 참으로 안타까울 뿐이다. 검사하는 어떤 방법을 썼다고 하더라도 테스트 한 후, 파일을 열기 전에 (다른 프로세스에 의해) 그 파일이 새로 만들어지거나 지워질 수 있기 때문이다. 이런 목적으로 쓸 수 있는 함수는 stat(), access(), fopen()이 있다. 물론 이 함수들 중에서는 fopen()이 가장 이식성이 뛰어나다. UNIX의 set-UID기능이 있다면 access() 함수는 주의깊게 써야 한다. 단순히 파일이 성공적으로 열렸다고 가정하는 것보다 항상 리턴 값을 검사해서 실패했는지 조사하는 것이 바람직하다.

13.5 Q&A

Q 54. 코드에서 잘못된 부분이 있나?

```
char c;
while ((c = getchar()) != EOF) ...
```

A. 일단, getchar의 리턴 값을 저장하는 변수는 반드시 int이어야 한다. EOF는 int 타입이기 때문에 이 리턴 값을 char에 저장하는 것은 EOF를 잘못 해석하게 할 소지가 있다. (특히 char의 타입이 unsigned인 경우 문제가 심각하다). 위의 코드처럼 getchar()의 리턴값을 char에 담을 경우, 두 가지 결과를 예상할 수 있다.

- char의 타입이 signed인 경우, 그리고 EOF가 -1로 정의된 경우, 문자값이 부호 확장(sign extension)되어, 문자값 255가 (C 언어 표현으로 '\377' 또는 '\xff') EOF와 같아진다. 따라서 입력 도중에 입력 처리가 끝나버릴 수 있다.
- char의 타입이 unsigned인 경우, EOF 값이 (상위 비트들이 잘라져 대개 255나 0xff의 값으로) 잘라져서, EOF로 인식이 되지 않아 버린다. 따라서 이 경우, 무한 루프에 빠져 버린다.

13.5 Q&A

Q 55. 프로그램의 프롬프트와 중간 단계까지의 출력은 (특히 파이프를 써서 출력을 다른 프로그램에 넘길 때), 때때로 스크린에 출력되지 않는다.

A. 입출력이 당장 스크린에 반영되기를 원한다면 (특히 텍스트가 `\n`으로 끝나지 않는 경우), `fflush(stdout)`을 불러 주는 것이 좋다. 대개의 경우 적절한 시기에 자동으로 `fflush()`를 불러주지만, 표준 출력이 인터렉티브(`interactive`)한 터미널로 연결되어 있다고 가정한 것이기 때문에, 가끔 제대로 동작하지 않을 수 있다.

13.6 실습 및 과제

- 실습 및 과제 진행
  - DCU Code : http://code.cu.ac.kr
  - 13 주차 실습
    - 코딩 13-1, 코딩 13-2, 코딩 13-3, 코딩 13-4 (p500-523)
  - 13 주차 과제
    - 11장 프로그래밍 연습 1-8번 (p529)

13.7 참고 문헌

- 컴퓨터 파일 참조 : https://ko.wikipedia.org/wiki/컴퓨터_파일
- C 파일 입출력 참조 : https://ko.wikipedia.org/wiki/C_파일_입출력

E N D