프로그래밍 기초

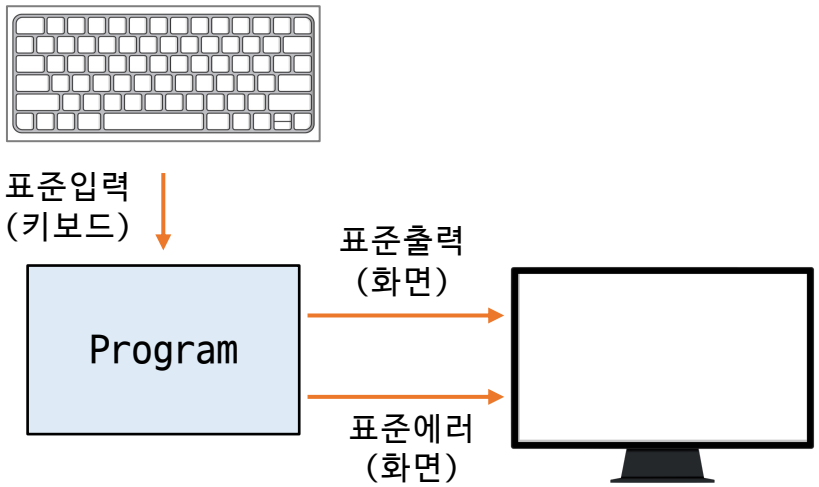
15장. 파일 처리 Part 1

학습목표

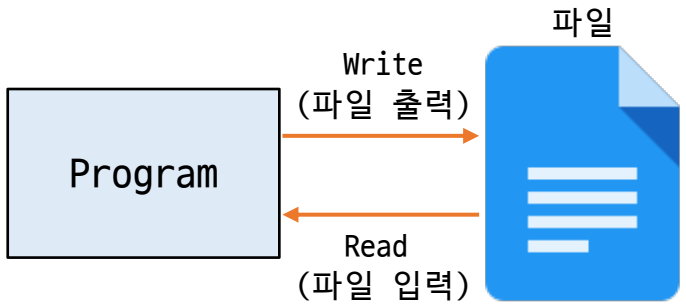
- **텍스트 파일과 이진 파일의 차이를 이해하고 설명할 수 있다.**
  - 주기억장치와 파일의 차이
  - 텍스트 파일과 이진 파일의 정의와 예
  - 파일 스트림과 파일모드의 이해
  - 함수 fopen()과 fopen_s(), fclose()의 사용
- **텍스트 파일의 입출력 함수를 이해하고 설명할 수 있다.**
  - 함수 fprintf()와 fscanf(), fscanf_s()의 사용
  - 함수 fputc()와 fgetc()의 사용
- **이진 파일의 입출력 함수를 이해하고 설명할 수 있다.**
  - 함수 fwrite()와 fread()의 사용
  - 함수 getw()와 putw()의 사용
- **파일 삭제 및 이름 바꾸기 함수를 이해하고 설명할 수 있다.**
  - 함수 remove()와 rename()의 사용

파일의 필요성

- **프로그램의 실행 결과 데이터를 파일로 직접 만들 수 있을까?**
  - 파일 출력: 프로그램이 생성하거나 처리한 데이터를 파일로 저장
  - 파일 입력: 파일에 저장된 데이터를 읽어서 프로그램이 사용



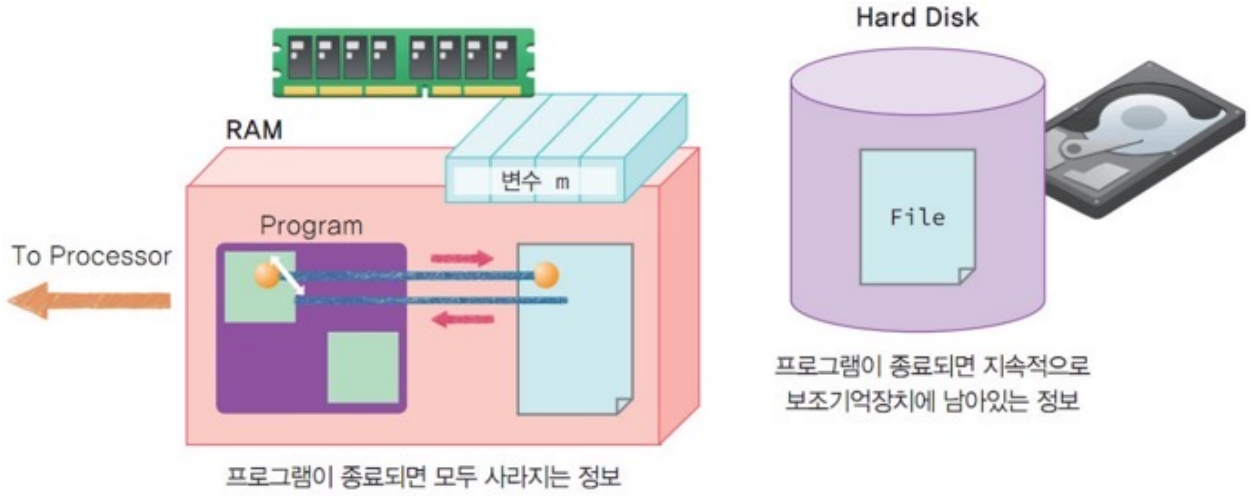
<표준 입출력>



<파일 입출력>

파일과 메모리

- **프로그램이 종료되면 모두 사라지는 자료**
  - 변수와 같이 프로그램의 내부에서 할당되어 사용되는 주기억장치의 메모리 공간
- **프로그램이 종료되더라도 계속 유지**
  - 보조기억장치인 디스크에 저장되는 파일(file)은 직접 삭제하지 않은 한 계속 남음
- **프로그램에서 사용하던 정보를 종료 후에도 계속 사용하고 싶다면**
  - 프로그램에서 파일에 그 내용을 저장

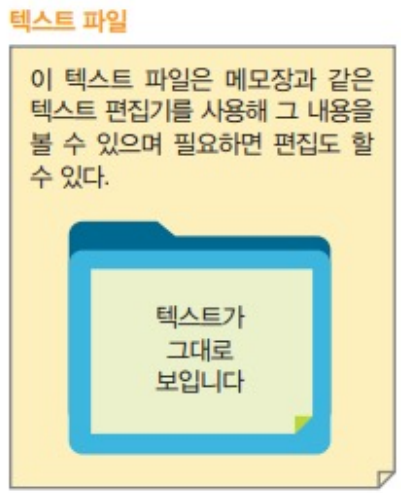


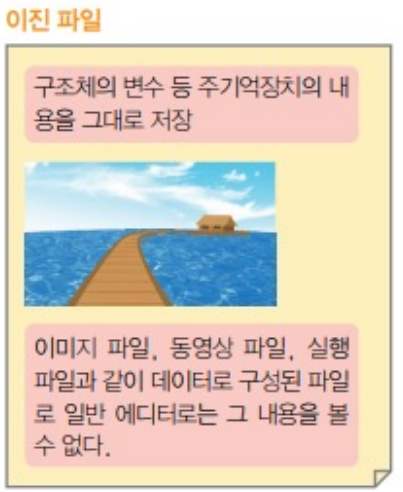
텍스트 파일과 이진파일

- **파일: 보조기억장치에 저장된 자료의 집합**
  - **텍스트 파일(text file)**과 **이진 파일(binary file)** 두 가지 유형으로 나뉨

- **텍스트 파일(Text file)**
  - 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(Binary file)**
  - 실행 파일, 그림 파일, 음악 파일, 동영상 파일 등
  - 목적에 알맞은 자료가 이진 형태(binary format)로 저장되는 파일
  - 입출력 속도도 텍스트 파일보다 빠름
    - ➢ 메모장과 같은 텍스트 편집기로는 그 내용을 볼 수 없음
    - ➢ 특정한 프로그램에 의해 실행

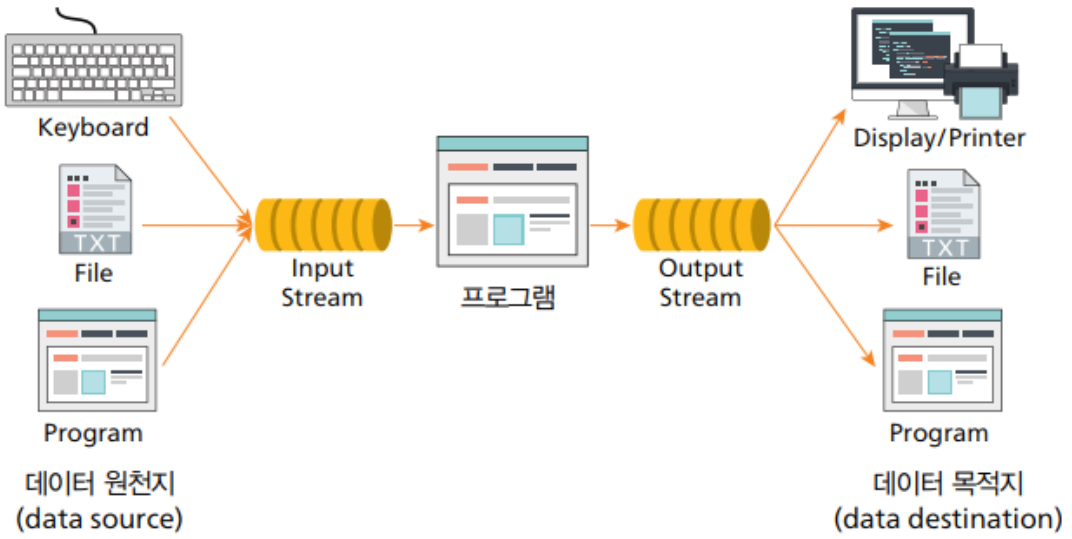




입출력 스트림 #1

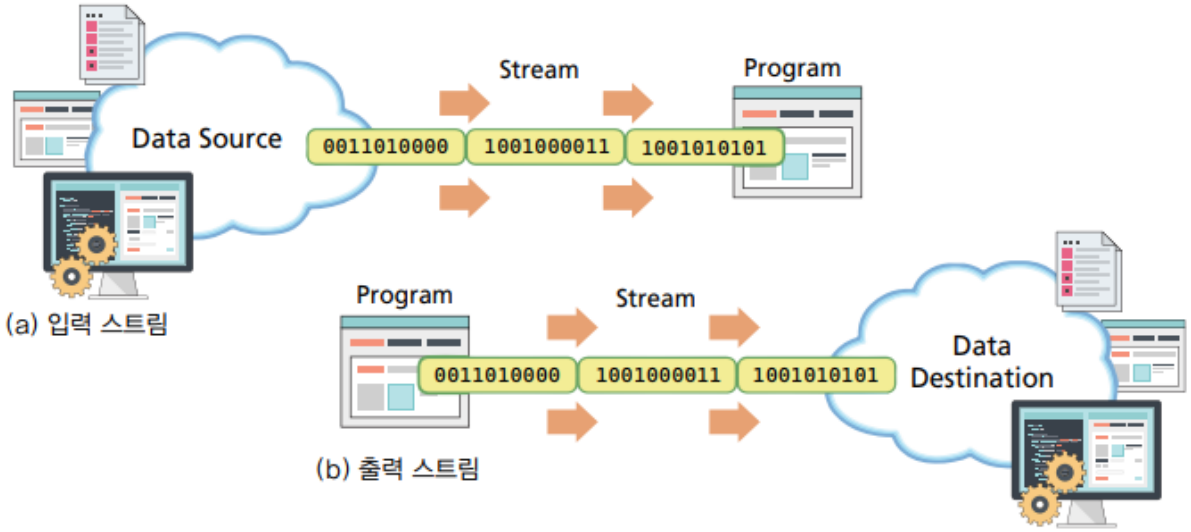
- **자료 입력과 출력은 자료의 이동으로 자료가 이동하려면 이동 경로가 필요**

- **입출력 스트림(io stream)**
  - 입출력 시 이동 통로
  - **표준 입력 스트림**: 키보드에서 프로그램으로 자료가 이동 경로
    - 함수 `scanf()`: 표준 입력 스트림에서 자료를 읽을 수 있는 함수
  - **표준 출력 스트림**: 프로그램에서 모니터의 콘솔로 자료가 이동 경로
    - 함수 `printf()`: 표준 출력 스트림으로 자료를 보낼 수 있는 함수



입출력 스트림(2)

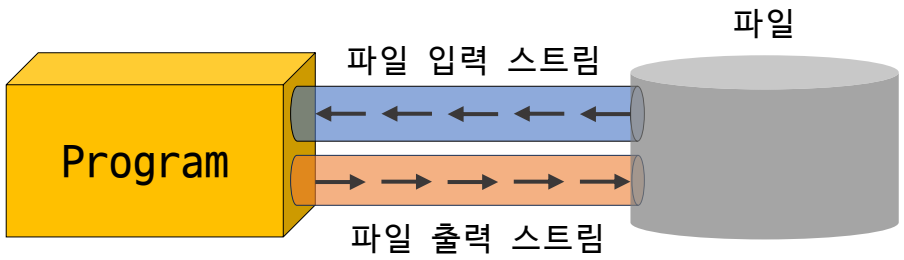
- **입력 스트림(input stream)**
  - 프로그램 내부로 데이터가 들어오는 경로
  - 표준 입력: 키보드를 통한 데이터 입력
  - 파일 입력: 파일로부터 자료를 읽는 것
  - 스크린 입력: 스크린에서 터치 정보
  - 네트워크 입력: 네트워크를 통해 자료가 전달



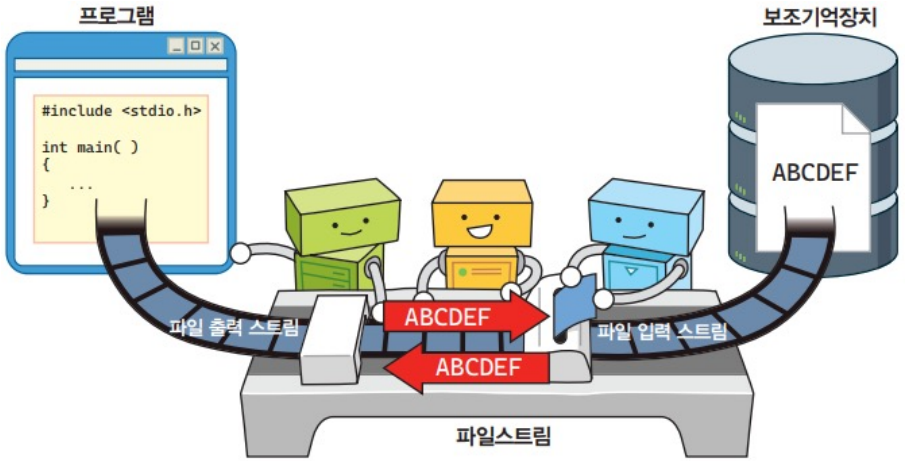
- **출력 스트림(input stream)**
  - 프로그램에서 데이터가 나가는 경로
  - 표준 출력: 목적지가 콘솔(화면)
  - 파일 출력: 파일에 원하는 데이터를 저장
  - 프린터 출력: 프린터에 출력물
  - 네트워크 출력: 네트워크로 출력이 되어 다른 곳으로 자료가 이동

파일 스트림 이해

- **파일 스트림**
  - 보조기억장치의 파일과 프로그램을 연결하는 전송 경로
  - 파일 입력 스트림(file input stream)
    - ➢ 파일에서 프로그램으로 자료의 입력을 위한 스트림
  - 파일 출력 스트림(file output stream)
    - ➢ 프로그램에서 파일로 출력을 위한 스트림



- **파일 스트림을 만들기 위해서는**
  - 특정한 파일 이름과 파일 모드가 필요
  - 파일 모드: 입력 또는 출력과 같은 스트림의 특성



함수 fopen(): 파일 스트림 열기

- **함수 fopen() 또는 fopen_s()를 이용**
  - 프로그램에서 특정한 파일과 파일 스트림을 연결하는 함수: 헤더 파일 stdio.h 필요
  - 현재 비주얼 스튜디오: fopen()는 fopen_s()로 대체, 함께 사용 가능

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

```
FILE *fopen_s(FILE ** streamptr, const char *filename, const char *mode); // Visual Studio
```

  - FILE 구조체: 함수 fopen()의 반환값: FILE *
  - 함수 fopen()은 인자가 파일이름과 파일열기 모드
    - ➢ 파일 스트림 연결에 성공하면, 파일 포인터를 반환
    - ➢ 실패하면, NULL을 반환

```
FILE* f; //파일 포인터
char* fname = "basic.txt"; //파일이름

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

함수 fopen_s(): 참고 (Visual Studio 전용)

- **함수 fopen_s(): Visual Studio 전용**
  - 첫 번째 인자: 파일 포인터의 주소값
  - 두 번째 인자: 문자열은 처리하려는 파일 이름
  - 세 번째 문자열: 파일 열기 종류인 모드
  - 리턴값
    - 파일 스트림 연결에 성공: 정수 0을 반환
    - 스트림 연결에 실패: 양수를 반환

```
FILE* f; //파일 포인터
char* fname = "basic.txt"; //파일이름

f = fopen_s(&f, fname, "w");
if (f == NULL)
{
    printf("파일이 열리지 않아 종료합니다.\n");
    exit(1);
};
```

함수 fopen()

- **파일 "basic.txt"를 여는 모듈**
  - 파일에 자료를 쓰기 위한 파일 스트림을 연결하기 위해서는 모드 값을 "w"로 기술

- **파일 열기 종류(모드)**
  - 텍스트 파일인 경우 "r", "w", "a" 등의 종류
  - 읽기모드(read): “r”
    - ➢ 읽기만 가능한 모드, 쓰기는 불가능
  - 쓰기모드(write): “w”
    - ➢ 쓰기만 가능한 모드, 읽기는 불가능
  - 추가모드(append): “a”
    - ➢ 파일 마지막에 추가적으로 쓰는 것만 가능한 모드
      - 읽기는 불가능
    - ➢ 파일에 쓰는 내용은 무조건 파일 마지막에 추가

```
FILE* f; //파일 포인터
char* fname = "basic.txt";

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

“w”: 파일 열기 모드

함수 fclose(): 파일 스트림 닫기

- **함수 fclose()**

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

  - 리턴값: 성공하면 0, 실패하면 EOF을 반환

- 파일 스트림을 연결한 후 파일 처리가 모두 끝났으면
  - fopen() 함수로 파일을 열었을 때 반환된 포인터 f 사용
  - fclose(f)를 호출하여 반드시 파일을 닫아야 함

- fclose() 기능
  - fopen()으로 연결한 파일 스트림을 닫는 기능을 수행
  - 내부적으로 파일 스트림 연결에 할당된 자원을 반납
  - 파일과 메모리 사이에 있던 버퍼의 내용을 모두 지우는 역할을 수행

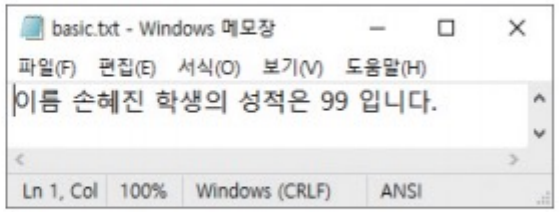
```
FILE* f; //파일 포인터
char* fname = "basic.txt";

if ((f = fopen(fname, "w")) == NULL) // 파일 열기
{
    printf("파일이 열리지 않아 종료합니다.\n");
    exit(1);
};
. . . // 파일 처리

fclose(f);  // 파일 닫기
```

이름과 성적 정보 파일로 저장

- **지정한 학생 이름과 점수를 파일 “basic.txt”에 출력**
  - 함수 fopen()와 fprintf()를 이용
  - 함수 fprintf(): 파일에 서식화된 문자열을 출력하는 함수

- **파일이 쓰기 모드로 열리지 않으면**
  - 함수 exit()를 이용하여 프로그램을 종료
    - 헤더 파일 #include <stdlib.h>가 필요
    - 함수를 강제로 종료하는 기능을 수행
    - 인자 값 0은 정상적인 종료를 의미: exit(0)
    - 0이 아닌 값은 정상적인 종료가 아님을 운영체제에게 알리는 의미로 사용

- **다음 예제 프로그램이 성공적으로 실행**
  - 일반적으로 소스 파일이 있는 폴더에 새로운 basic.txt 파일이 생성
    - 텍스트 파일이므로 모든 편집기로 볼 수 있음
    - 메모장으로 파일 basic.txt를 열어 확인



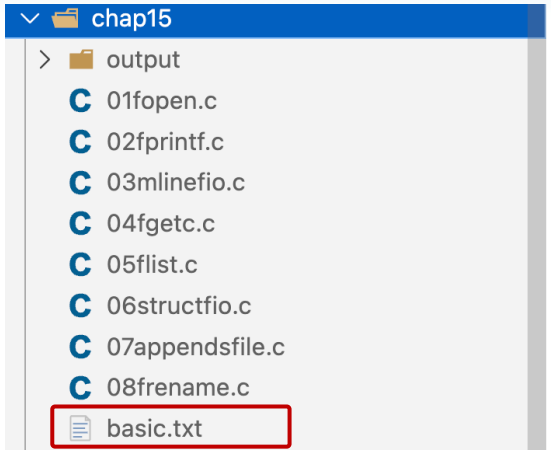
이름과 성적 정보 파일로 저장 예제 (실습)

<01fopen.c>

```c
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h> //for exit()

int main()
{
    char* fname = "basic.txt"; //파일이름
    FILE* f; //파일 포인터

    char name[30] = "손혜진"; // 파일에 쓰려는 자료
    int point = 99;
    //if (fopen_s(&f, fname, "w") != 0)
    if ((f = fopen(fname, "w")) == NULL)
    {
        printf("파일이 열리지 않아 종료합니다.\n");
        exit(1);
    }
    //파일 "basic.txt"에 쓰기
    fprintf(f, "이름 %s 학생의 성적은 %d 입니다.\n", name, point);
    fclose(f);

    //표준출력 콘솔에 쓰기
    printf("이름 %s 학생의 성적은 %d 입니다.\n", name, point);
    puts("프로젝트 폴더에서 파일 basic.txt를 메모장으로 열어 보세요.");

    return 0;
}
```

Visual Studio에서 fopen() 사용시 필요

서식화된 파일 출력(저장)
- 지정된 파일에 문자열 저장



실행 결과

```
이름 손혜진 학생의 성적은 99 입니다.
프로젝트 폴더에서 파일 basic.txt를 메모장으로 열어 보세요.
```

LAB 파일 “myinfo.txt”을 열어 간단한 정보를 저장

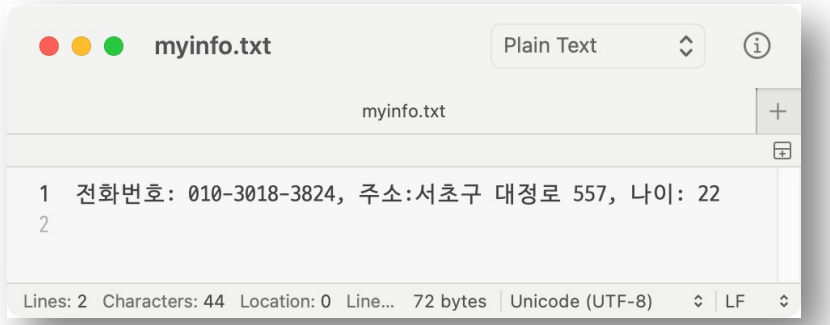
- **파일 “myinfo.txt”를 쓰기 모드로 열어**
  - 전화번호, 주소, 나이 등의 간단한 정보를 저장
- **함수 fprintf(f, …)**
  - 첫 인자에 파일 포인터를 대입하면 파일에 서식화된 문자열을 출력(저장)
- **파일 f를 닫기**
  - fclose(f)
- **파일 저장 내용**
  - 프로젝트 폴더
    - ➢ 파일 myinfo.txt를 메모장으로 열기
  - **전화번호: 010-3018-3824, 주소:서초구 대정로 557, 나이: 22**

Lab 예제 lab1basicfio.c

```c
#include <stdio.h>
#include <stdlib.h> //for exit()

int main()
{
    FILE* f; //파일 포인터

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

    //파일에 쓰려는 자료
    char tel[15] = "010-3018-3824";
    char add[30] = "서초구 대정로 557";
    int age = 22;
    //파일 " myinfo.txt"에 형식화된 문자열 저장(쓰기)
    fprintf(f, "전화번호: %s, 주소:%s, 나이: %d\n", tel, add, age);
    fclose(f); //파일 닫기

    //표준출력 콘솔에 쓰기
    printf("전화번호: %s, 주소:%s, 나이: %d\n", tel, add, age);
    puts("프로젝트 폴더에서 파일 myinfo.txt를 메모장으로 열어 보세요.");

    return 0;
}
```

fprintf() 함수
- 서식화된 파일 출력

printf() 함수
- 서식화된 화면 출력




```
전화번호: 010-3018-3824, 주소:서초구 대정로 557, 나이: 22
프로젝트 폴더에서 파일 myinfo.txt를 메모장으로 열어 보세요.
```

파일에 서식화된 문자열 입출력

- **fprintf(), fscanf(), fscanf_s() 이용**
  - 텍스트 파일에 정해진 형식으로 자료를 쓰거나 읽기 위한 함수
    - 첫 번째 인자는 입출력에 이용될 파일 포인터: fprintf(FILE *file, ...)
    - 두 번째 인자는 입출력에 이용되는 제어 문자열: (..., "%s %c %d", ...)
    - 세 번째 인자들은 입출력 될 변수 또는 상수 목록
  - 첫 번째 인자에 각각 stdin, stdout을 이용하면 표준 입력, 표준 출력으로 사용 가능

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

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

- **기호 상수 stdin, stdout, stderr**
  - 헤더 파일 stdio.h에 정의되어 있는 값

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

파일 쓰기 및 파일 읽기 프로그램을 작성

- **쓰기 모드로 파일 열기**
  - 표준 입력으로 받은 학생 이름과 중간 점수, 기말 점수를 파일에 기록하고 파일을 닫기
  - 다시 읽기 모드로 그 파일을 열기
    - ➢ 기록된 내용을 읽어서 표준출력으로 출력하는 프로그램

- **함수 scanf(): 표준입력을 빈칸으로 구분하므로**
  - 빈칸이 없는 이름을 입력

- **함수 fprintf(fp, ...): 파일 f에 출력**
  - 학생이름, 중간점수, 기말점수
  - 변수 cnt는 이름 앞에 번호를 붙이려는 목적으로 이용

- **파일 f에서 함수 fscanf(fp, ...)를 이용**
  - cnt, 학생이름, 중간, 기말이므로 다음과 같은 순서로 파일에서 읽음

실습 예제 15-2: 02fprintf.c (실습)

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

int main()
{
    char fname[] = "grade.txt";
    FILE *f;
    char name[30];
    int point1, point2, cnt = 0;

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

    printf("이름과 성적(중간, 기말)을 입력하세요.\n");
    scanf("%s %d %d", name, &point1, &point2);

    fprintf(f, "%d %s %d %d\n", ++cnt, name, point1, point2);

    fclose(f);
```

파일 쓰기 모드 열기("w")
↓
화면 입력 (scanf)
↓
파일 저장 (fprintf)

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

    // 파일"grade.txt"에서 읽기

    fscanf(f, "%d %s %d %d\n", &cnt, name, &point1, &point2);

    // 표준 출력(화면)에 쓰기
    fprintf(stdout, "%-4s %-10s %-10s %-8s\n",
                    "번호", "이름", "중간", "기말");
    fprintf(stdout, "%-4d %-12s %-8d %-8d\n",
                    cnt, name, point1, point2);
    fclose(f);
    return 0;
}
```

파일 읽기 모드 열기("r")
↓
파일 읽기(fscanf)
↓
화면 출력 (fprintf)

실행 결과

```
이름과 성적(중간, 기말)을 입력하세요.
이재현 100 90

번호 이름       중간      기말
1    이재현     100       90
```

함수 fgets()와 fputs()

- **함수 fgets()**

```
char* fgets(char *buf, int maxsize, FILE *file);
```

  - 파일로부터 한 행의 문자열을 읽어오는 함수
  - maxsize-1 까지 읽지 않은 경우: 개행 문자(\n)를 만나면
    - ➢ 개행 문자까지 읽고, 마지막에 NULL 문자 ('\0')를 추가함
  - maxsize-1 까지 읽은 경우: 개행 문자를 만나지 않았더라도 읽기를 멈춤
    - ➢ 마지막에 NULL('\0') 자동 추가
  - 파일의 끝(EOF)에 도달한 경우, 읽기를 멈춤

- **함수 fputs()**

```
int fputs(char *buf, FILE *file);
```

  - 한 행의 문자열을 파일로 저장하는 함수
  - 자동으로 개행 문자('\n') 추가 하지 않음
    - ➢ 개행 문자가 필요한 경우, 명시적으로 추가해야 됨: fputs("abcdef\n", fp);

함수 feof()와 ferror()

- **함수 feof()**

```
int feof(FILE *file);
```

  - 파일 스트림의 EOF(End Of File) 표시를 검사하는 함수
  - 파일의 끝(EOF)에 도달했으면, 0이 아닌 값(1)을 반환
  - 읽기 작업이 파일의 끝에 도달하지 못했으면, 0을 반환

- **함수 ferror()**

```
int ferror(FILE *file);
```

  - 파일 처리에서 오류가 발생했는지 검사하는 함수
  - 리턴값
    - ➢ 파일 처리에서 오류가 발생하면, 0이 아닌 값을 반환
    - ➢ 오류가 발생하지 않으면, 0을 반환

```
FILE* f; //파일 포인터
char* fname = "basic.txt";

while(!feof(stdin))
{
    . . .
    fgets(name_score, 80, stdin);
};
```

while(feof(stdin) == 0)

실습 예제 15-3: 여러 줄의 성적 입력 및 저장

- **표준 입력으로 여러 줄을 입력 받아 여러 줄을 파일 grade.txt에 출력**
  - 함수 fgets()와 fputs()를 이용
  - 여러 줄의 표준입력을 처리
    - while (!feof(stdin)) {…} 구문을 이용
    - 함수 fprintf()를 이용하여 줄 번호를 출력
    - 맨 앞에 1부터 순차적으로 번호를 삽입

- **표준입력에서 입력을 종료**
  - 파일의 끝(EOF)을 의미하는 키 Ctrl + Z를 새로운 행의 처음에 누름
  - Windows: Ctrl + Z
  - Linux, Mac: Ctrl + D

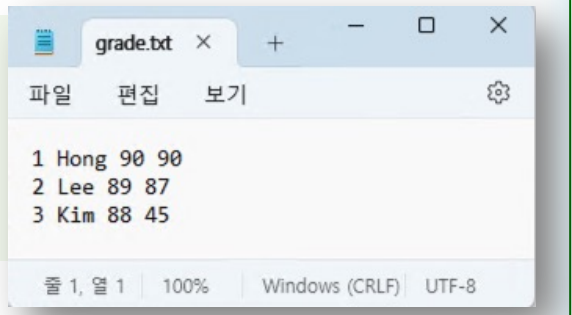
실습 예제 15-3: 여러 줄의 성적 입력 및 저장 (03mlinefio.c)

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

int main()
{
    char fname[] = "grade.txt";
    char name_score[80] = {0};
    int cnt = 0;
    FILE* f;

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

    printf("이름과 성적(중간, 기말)을 입력하세요.\n");
    fgets(name_score, 80, stdin);
```

화면 입력 과정에서
개행문자('\n')이 포함됨

```c
    //콘솔에 이름 중간 기말 입력하고 Enter 치고
    //계속 여러 줄에 여러 학생의 성적을 입력하다가
    //종료하고 싶을 때 새로운 줄 처음에 ctrl + Z 누르고 Enter 종료

    while (!feof(stdin)) // Ctrl + Z 표준입력의 종료 확인
    {
        fprintf(f, "%d ", ++cnt); //맨 앞에 번호를 삽입
        fputs(name_score, f); // 입력 받은 이름과 성적 2개 저장

        fgets(name_score, 80, stdin); //표준입력('\n'이 문자열에 저장)
    }

    fclose(f);

    return 0;
}
```

```
이름과 성적(중간, 기말)을 입력하세요.
Hong 89 98
Lee 92 95
Kim 88 92
^Z
```



함수 fgetc()와 fputc()

- **함수 fgetc()**

```
int fgetc(FILE *file);
```

  - 파일로부터 문자 하나를 읽어오는 함수
    - ➢ getc(FILE *file)와 동일 기능
  - 파일의 끝에 도달하면 EOF(-1)를 리턴

- **함수 fputc()**

```
int fputc(int ch, FILE *file);
```

  - 문자 하나를 파일로 저장하는 함수
    - ➢ putc(int ch, FILE *file)와 동일 기능
  - 함수들은 문자 하나의 입출력의 대상인 파일 포인터를 인자로 이용

실습 예제 15-4

- **여러 문자를 표준입력으로 받아서 파일 char.txt에 저장한 다음, 다시 파일에서 문자를 읽어서 표준 출력하는 프로그램**

- **프로그램 실행**
  - 프로젝트 폴더에 파일 char.txt 생성
  - 'x'를 입력하기 전까지의 문자가 입력되고, char.txt에 저장
    - `int ch = getchar();`
    - `fputc(ch, fp);`
  - 'x'가 입력되면 파일 close()

  - char.txt 파일을 읽기 모드로 열고 한 글자씩 읽어서 화면에 출력
    - `int ch = fgetc(f);`
    - `putchar(ch);`

실습 예제 15-4: 04fgetc.c

<04fgets.c>

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

int main()
{
    char fname[] = "char.txt";
    FILE *f; // 파일 포인터

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

    puts("영어 문자를 계속 입력하다가 종료하려면 x를 입력 >>");

    int ch; // 입력된 문자 저장
    // 파일 "char.txt"에 문자 저장(출력)
    while ((ch = getchar()) != 'x')
        fputc(ch, f);

    fclose(f);
    puts("");
    if (fopen(fname, "r") == NULL) // 읽기모드로 파일 열기
    {
        printf("파일이 열리지 않습니다.\n");
        exit(1);
    }
    /* 파일 char.txt에서 문자를 읽고 콘솔에 출력하는 부분 */
    while ((ch = fgetc(f)) != EOF)
        putchar(ch);

    fclose(f);
    puts("");
    return 0;
}
```

실행 결과

```
영어 문자를 계속 입력하다가 종료하려면 x를 입력 >>
Hello Java Python Enter
C language Enter
Kotlin x Enter

Hello Java Python
C language
Kotlin
```

chap15 > char.txt

```
1  Hello Java Python
2  C language
3  Kotlin
```

실습 예제 15-5: 파일 내용을 표준출력으로 그대로 출력 (실습)

<05flist.c>

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

int main(void)
{
    FILE *f;
    if ((f = fopen("05flist.c", "r")) == NULL)
    {
        printf("파일이 열리지 않습니다.\n");
        exit(1);
    }

    int ch, cnt = 0; // 문자를 저장할 ch, 행번호를 저장할 cnt
    printf("%4d: ", ++cnt); // 1행 처음에 번호 1 출력
    while ((ch = fgetc(f)) != EOF)
    {
        putchar(ch); // putc(ch, stdout);
        if (ch == '\n') // '\n'을 만나면 행 처음에 행 번호 출력
            printf("%4d: ", ++cnt);
    }
    printf("\n");
    fclose(f);

    return 0;
}
```

실행 결과

```
 1: #include <stdio.h>
 2: #include <stdlib.h>
 3:
 4: int main(void)
 5: {
 6:   FILE *f;
 7:   // if (fopen_s(&f, "05flist.c", "r") != 0) //읽기 모드로 파일 열기
 8:   if ((f = fopen("05flist.c", "r")) == NULL)
 9:   {
10:           printf("파일이 열리지 않습니다.\n");
11:           exit(1);
12:   }
13:
14:   int ch, cnt = 0;         // 문자를 저장할 ch, 행번호를 저장할 cnt
15:   printf("%4d: ", ++cnt); // 1행 처음에 번호 1 출력
16:   while ((ch = fgetc(f)) != EOF)
17:   {
18:           putchar(ch);    // putc(ch, stdout);
19:           if (ch == '\n') // 2행부터 행 처음에 행 번호 출력
20:                   printf("%4d: ", ++cnt);
21:   }
22:   printf("\n");
23:   fclose(f);
24:
25:   return 0;
26: }
27:
```

LAB 파일의 대소문자를 서로 바꿔 다른 파일로 생성

- **소스 파일 lab2uplowercher.c 의 알파벳에서**
  - 알파벳 대문자는 소문자로, 소문자는 대문자로 변환
    - 파일 convertchar.c에 저장하는 프로그램
  - 두 파일을 각각 열기 모드와 쓰기 모드로 열고
    - 함수 fgetc()로 문자 하나를 읽어옴
    - 알파벳 대문자는 소문자로, 소문자는 대문자로 변경
      - isalpha(ch)
      - isupper(ch), islower(ch)
      - toupper(ch), tolower(ch)
    - 함수 fputc()로 변환된 문자를 convertchar.c에 저장
  - 문자 처리에 오류가 발생해 소스에서 한글은 사용 안함
- **결과**
  - File convertchar.c is created!!!

convertchar.c (~/workspace_cprog/chap15) - VIM

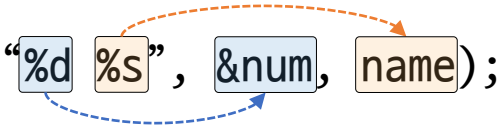
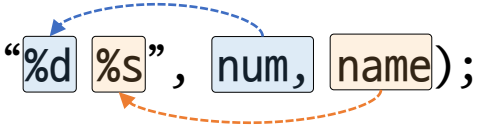
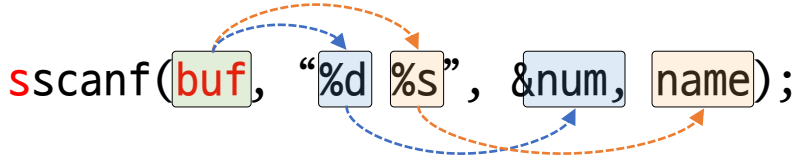
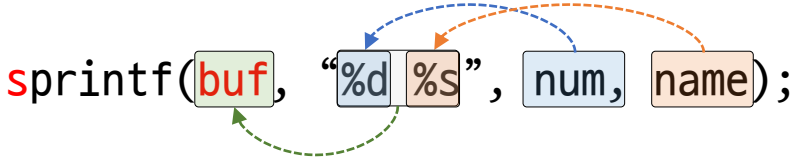
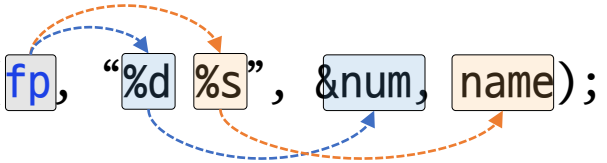
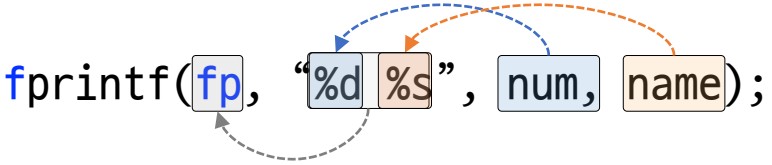
```
#DEFINE _crt_secure_no_warnings
#INCLUDE <STDIO.H>
#INCLUDE <STDLIB.H>
#INCLUDE <CTYPE.H>

INT MAIN(VOID)
{
    file *F1, *F2;
    IF ((F1 = FOPEN("LAB2UPLOWERCHAR.C", "R")) == null)
    {
        PRINTF("CANNOT OPEN THIS FILE\N");
        EXIT(1);
    }
    IF ((F2 = FOPEN("CONVERTCHAR.C", "W")) == null)
    {
        PRINTF("CANNOT OPEN THIS FILE\N");
        FCLOSE(F1);
        EXIT(1);
    }

    CHAR A;
    WHILE ((A = GETC(F1)) != eof) // FGETC(F1)
    {
        IF (ISALPHA(A))
            IF(ISLOWER(A))
                A = TOUPPER(A);
            ELSE IF (ISUPPER(A))
                A = TOLOWER(A);
        PUTC(A, F2); // FPUTC(A, F2)
    }

    FCLOSE(F1);
    FCLOSE(F2);
    PRINTF("fILE CONVERTCHAR.C IS CREATED!!!\N");

    RETURN 0;
}
~
~
```

LAB. lab2uplowerchar.c (실습)

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

int main(void)
{
    FILE *f1, *f2;
    if ((f1 = fopen("lab2uplowerchar.c", "r")) == NULL)
    {
        printf("cannot open this file\n");
        exit(1);
    }

    if ((f2 = fopen("convertchar.c", "w")) == NULL)
    {
        printf("cannot open this file\n");
        fclose(f1);
        exit(1);
    }
    char a;
    while ((a = fgetc(f1)) != EOF)
    {
        if (isalpha(a))
            if(islower(a))
                a = toupper(a);
            else if (isupper(a))
                a = tolower(a);
        putc(a, f2); // fputc(a, f2)
    }

    fclose(f1);
    fclose(f2);
    printf("File convertchar.c is created!!!\n");

    return 0;
}
```

형식화된 입출력 함수 정리

- **형식화된 화면, 문자열, 파일 입출력**

	읽기(입력)	쓰기(출력)
화면 입출력		
문자열 입출력 (string)		
파일 입출력 (file)		

문자 단위 파일 입출력: fputc(), fgetc()

- **문자 단위(char) 파일 쓰기 및 읽기**
  - fputc(): 한 글자씩 파일로 쓰기
  - fgetc(): 파일에서 한 글자씩 읽기

라인 단위 파일 입출력: fpus(), fgets()

- **한 줄(line) 단위 파일 쓰기 및 읽기**
  - fputs(buf, fp): 자동으로 개행 문자('\n') 추가 하지 않음
    - 개행 문자가 필요한 경우, 명시적으로 추가해야 됨: fputs("abcdef\n", fp);
    - puts('문자열'): 화면 출력 시 자동으로 개행 문자 추가
  - fgets(buf, size, fp)
    - 1. 개행 문자를 만난 경우: 개행 문자까지 읽고 buf에 저장
    - 2. size-1 까지 읽은 경우: 개행 문자를 만나지 않았더라도 읽기 중단(마지막에 NULL('\0') 자동 추가)
    - 3. 파일의 끝(EOF)에 도달한 경우

Questions?