
컴퓨터 프로그래밍 I

(CSE2003-3)

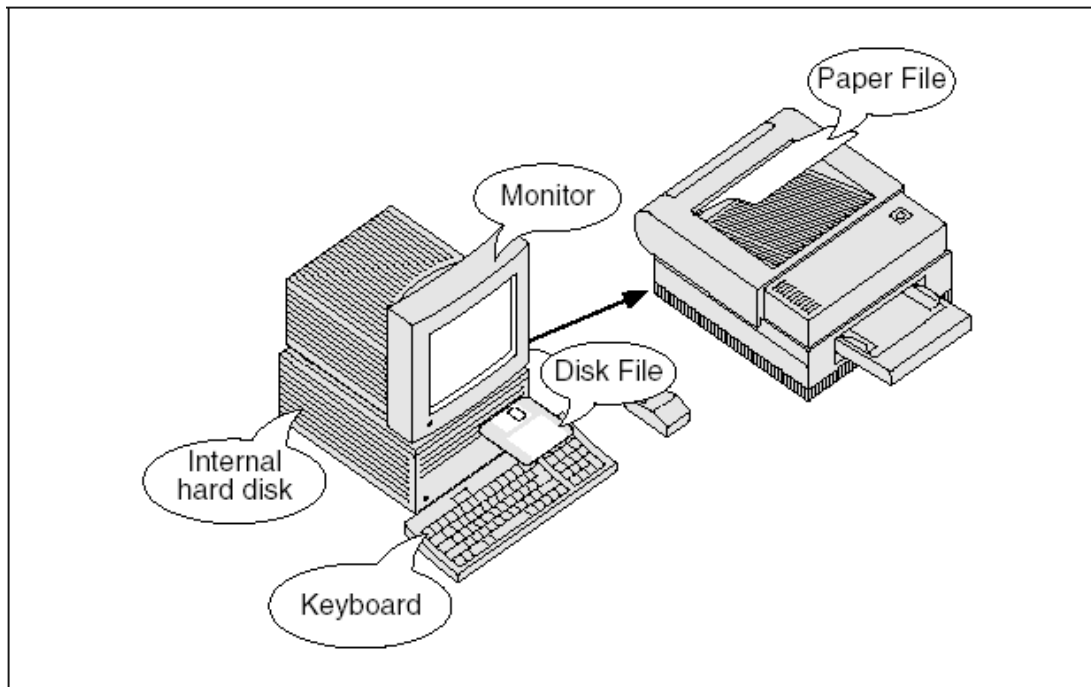
Mon/Wed 16:30-17:45 pm
Lecture 18

Text File Input/Output

Files

◆ File(파일)

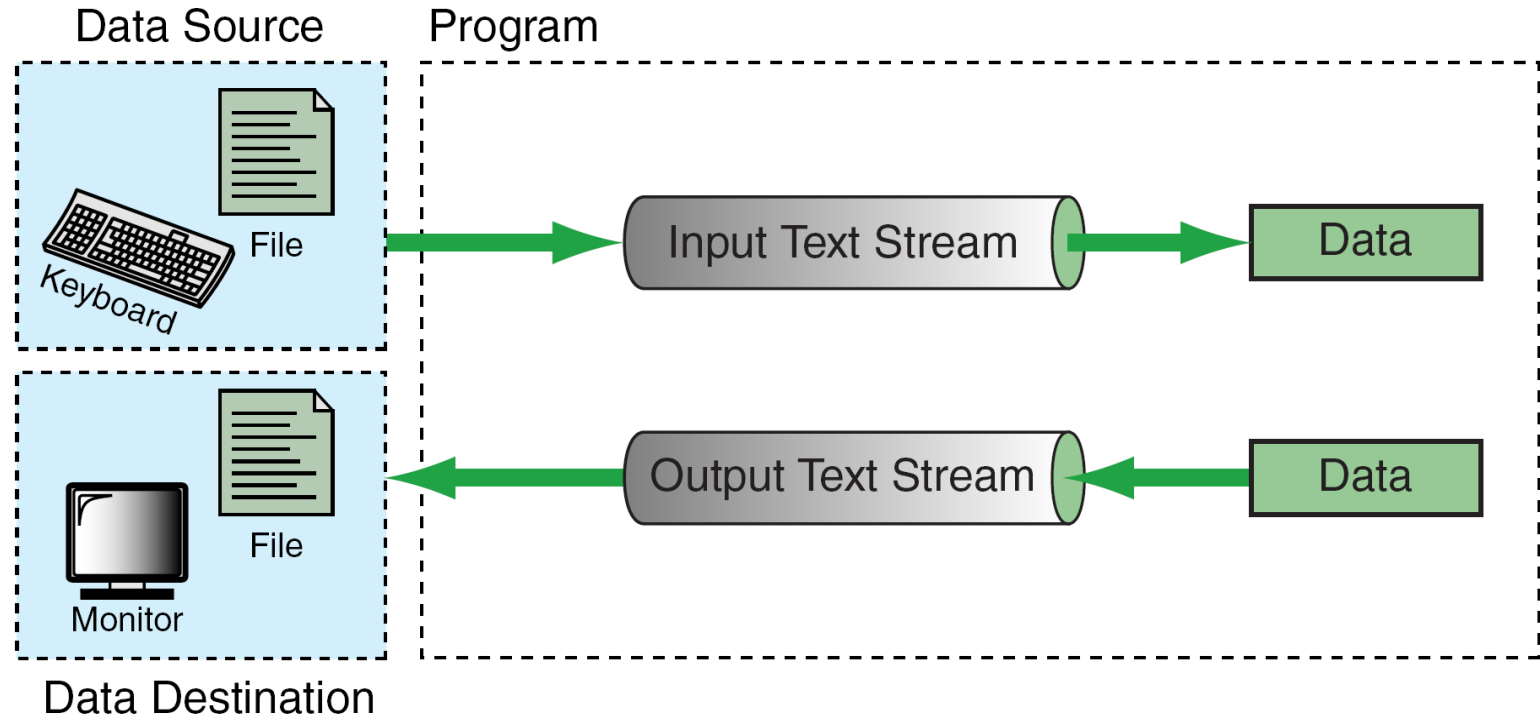
- 하나의 단위로 취급하는 external data(외부 데이터)의 단위
 - ◆ C에서, 파일(file)은 디스크 파일에서 터미널 또는 프린터에 이르기까지 어떤 것도 될 수 있음



Streams

◆ Stream(스트림)

- 데이터의 source와 destination은 file이지만, data는 스트림(stream)을 통해 입력되고 스트림을 통해 출력된다.



Streams

◆ 스트림의 종류

- Text stream(텍스트 스트림)
 - ◆ 연속된 문자들로 구성되어 있다.
 - ◆ line(레코드) 단위로 분리 (\n : new-line 특수문자로 분리)
- Binary stream(이진 스트림)
 - ◆ 정수, 실수 등의 연속된 데이터 값들로 이루어져 있다.
 - ◆ Text stream과 달리 메모리에 표현된 것들을 변환없이 전송하므로 속도가 빠르다.

◆ 스트림-파일을 처리하는 4단계

- 1) Creating a Stream
- 2) Opening a File
- 3) Using the Stream Name
- 4) Closing the Stream

Streams

◆ Creating a Stream

- 스트림을 선언하면 스트림이 생성된다.
- FILE 타입은 파일을 읽고 쓰는데 필요한 정보들을 갖고 있다.
- FILE 뒤의 *(asterisk)
 - ◆ spData가 스트림의 주소를 갖고 있는 포인터변수임을 의미

```
FILE* spData;
```

◆ Opening a File

- 특정 스트림과 파일을 연관 시킴
- 파일이 열리면 파일과 프로그램 사이에서 정보가 교환될 수 있다.

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

첫 번째 인자는 처리하려는
파일 이름을 지정하는 인자

두 번째 인자는 파일 처리의
종류를 지정하는 인자

Streams

◆ Using the Stream Name

- 스트림을 생성한 뒤에는 해당 스트림 포인터(spData)를 사용하여 대응하는 파일을 access(접근) 할 수 있으며, 이 스트림 포인터는 모든 함수에서 사용 가능하다.

◆ Closing the Stream

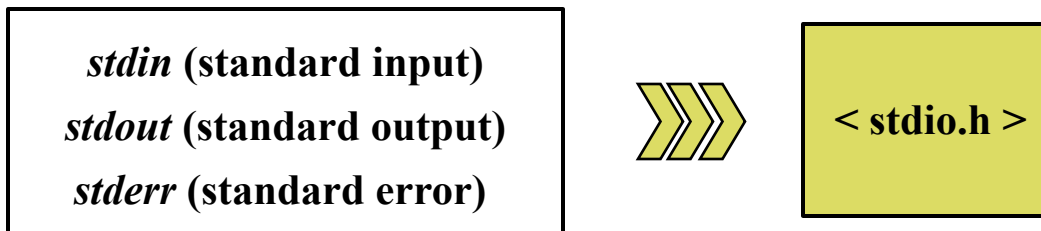
- 파일과 스트림의 연결을 끊는다.

```
fclose(FILE * stream)
```

Streams

◆ System-Created Streams

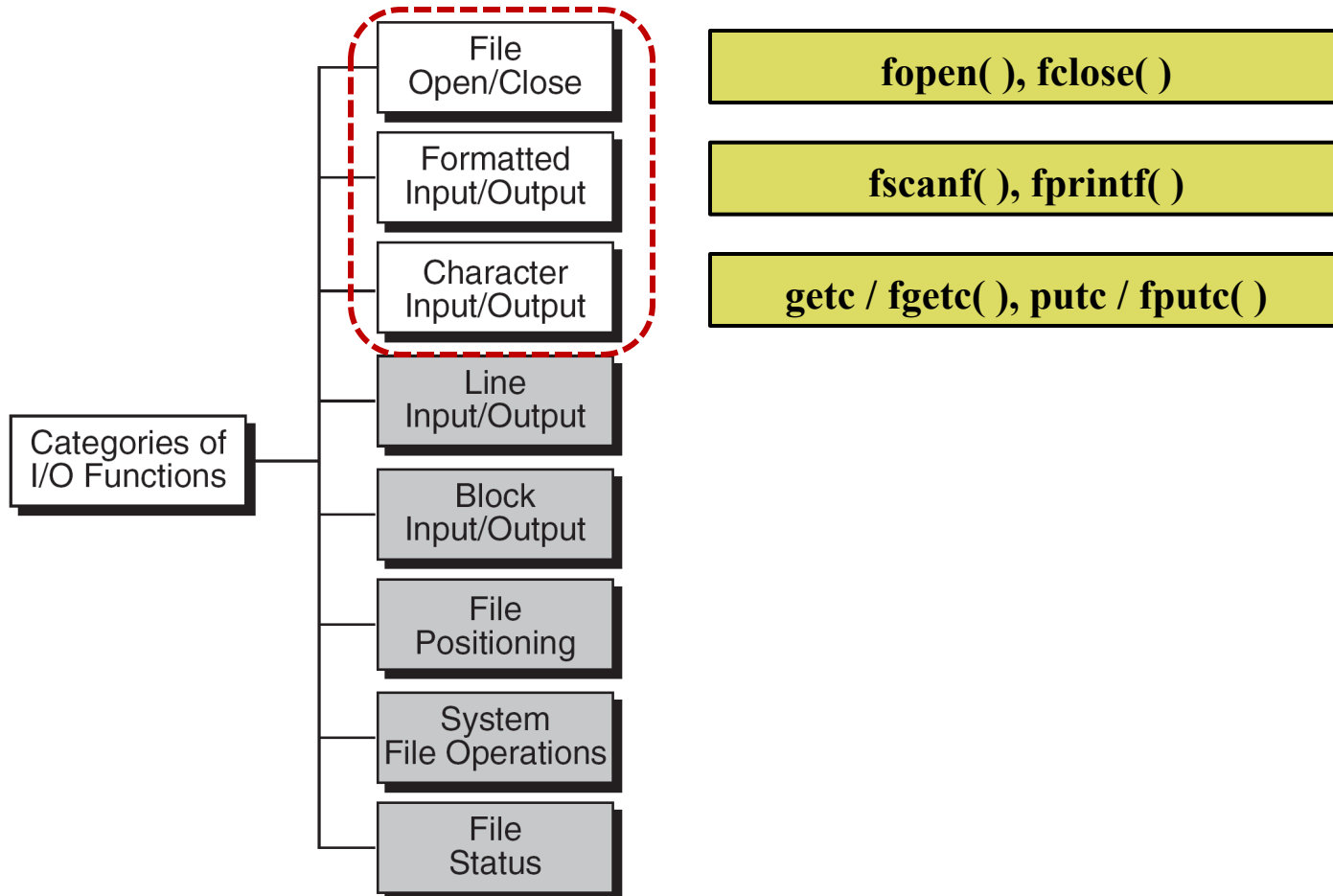
- C 언어는 터미널(keyboard or monitor)과의 의사소통을 위해 표준 스트림(standard stream)을 제공한다.



- 표준 스트림은 사용자가 열거나 닫을 필요가 없다.
 - ◆ 운영체제에 의해서 자동으로 처리됨
- C는 키보드에서 데이터를 입력받고 모니터로 출력을 하기 위해서 표준 스트림을 사용하는 많은 표준 함수(standard function)를 갖고 있다.
 - ◆ *ex) printf, scanf, etc.*

Standard Library Input/Output Functions

◆ standard input/output 함수들의 타입들



파일 처리에 관련한 여러 함수들

◆ ANSI C 파일 시스템은 여러 가지 상호 연관된 함수들로 구성

함수 이름	기능
fopen()	파일 열기
fclose()	파일 닫기
putc()	파일에 문자 쓰기
fputc()	putc()와 같은 것
getc()	파일로부터 문자 읽기
fgetc()	getc()와 같은 것
fseek()	파일에서 지정된 바이트 찾기
fprintf()	콘솔을 위한 printf()를 파일에 적용한 것
fscanf()	콘솔을 위한 scanf()를 파일에 적용한 것
feof()	파일의 끝에 도달하면 참을 리턴.
ferror()	오류가 발생하면 참을 리턴
rewind()	파일 위치 지시자를 파일의 시작점으로 재설정
remove()	파일 삭제하기
fflush()	파일버퍼 비우기

* ANSI(American National Standard Institute) : 미국표준협회.

파일의 열기

◆ 파일의 열기와 닫기

● 파일 열기

- ◆ 파일을 열기 위해선 *fopen* 함수를 쓴다.

```
fopen("filename", "mode");
```

- ◆ filename : 파일의 이름과 경로 정보를 지니는 문자열
- ◆ mode : 파일을 어떻게 사용할 것인지를 C에게 알려주는 문자열
파일의 특징 및 용도를 결정짓는다
- 파일명만 전달하는 경우, 현재 디렉토리에서 전달된 이름의 파일을 찾아서 개방한다. 현재 디렉토리가 아닌 다른 곳에 있는 파일을 오픈하기 위해서는 반드시 파일의 full 경로를 포함한 이름을 사용해야 한다.

```
fpTempData = fopen("TEMPS.DAT", "w");  
fpTempData = fopen("A:\\\\TEMPS.DAT", "w");
```

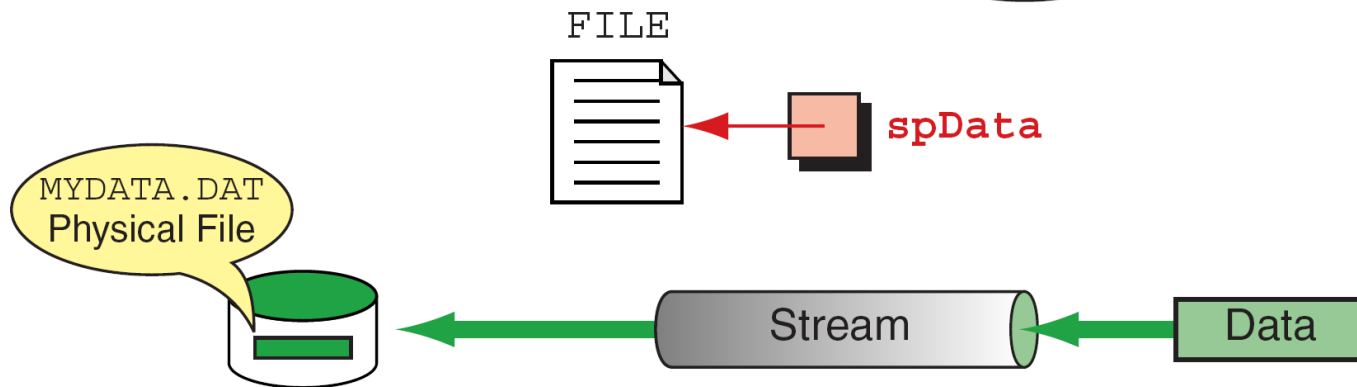
파일의 열기

◆ 파일 열기의 결과

```
#include <stdio.h>
...
{
  int main (void)
  {
    FILE* spData;
    ...
    spData = fopen("MYDATA.DAT", "w");
    ...
  } // main
```

Internal
File Variable

External
File Name



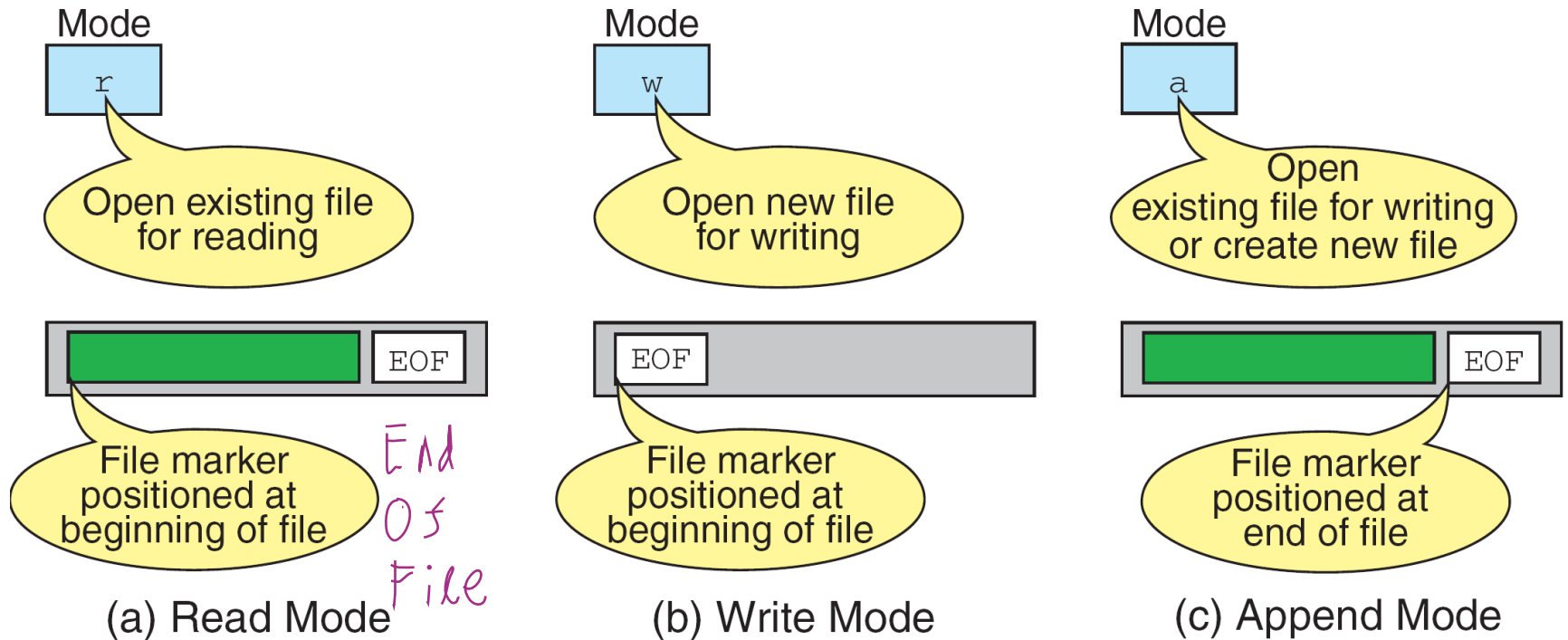
파일 처리 모드

◆ 파일 처리 모드 종류와 의미

모드	의 미
r	읽기(read) 모드로 파일을 개방한다. 파일이 있으면 마커(marker)가 처음에 위치한다. 파일이 없으면 널(NULL) 포인터를 리턴한다.
w	쓰기(write) 모드로 파일을 개방한다. 기존 파일이 있으면 지워진다. 파일이 없으면 새로운 파일을 생성한다.
a	추가(append) 모드로 파일을 개방한다. 기존 파일이 있으면 마커가 끝에 위치한다. 파일이 없으면 새로운 파일을 생성한다.
r+	파일을 읽고 쓰기 위해 개방한다. 파일이 없으면 새로운 파일을 생성한다. 파일이 있으면, 원래 존재하는 파일의 데이터를 덮어쓰게 된다.

파일 처리 모드

◆ 파일 처리 모드



파일의 닫기

■ 파일 닫기

- 파일이 더 이상 필요 없을 시에는 파일을 닫아주고 buffer space와 같은 resource를 시스템에 돌려주어야 한다.
- 파일에 대한 쓰기, 읽기 등의 작업이 종료된 후, `fclose()`로 파일을 닫아주는 것이다.
- 해당 파일을 성공적으로 닫았을 때는 0을 리턴, 오류가 발생하면 -1을 리턴한다.

```
FILE* fp;           // 파일포인터 변수
int res;            // fclose 함수의 리턴값 저장
fp = fopen("a.txt", "r"); // 파일 개방
...
res = fclose(fp);    // 파일 종결
if(res != 0) {
    printf("파일이 닫히지 않았습니다.\n");
    return 1;
}
```

파일의 열기와 닫기 에러

◆ 파일의 열기와 닫기 에러

- 파일을 열 때
 - ◆ 열려고 하는 파일이름이 디스크에 존재하지 않을 때
 - ◆ 새로운 파일을 생성할 공간이 디스크에 없을 때
 - ◆ 실패할 경우 스트림 포인터 변수는 NULL값을 갖게 된다.
- 파일을 닫을 때
 - ◆ *fclose* 함수는 파일이 성공적으로 닫히면 0을 리턴한다.
 - ◆ 에러가 있을 경우 EOF를 리턴한다.
- ***if* 문을 사용하여 항상 파일이 성공적으로 열리고 닫혔는지 확인한다!!!**

파일의 열기와 닫기

◆ 예제 프로그램 - 파일 열기와 닫기

```
1 #include<stdio.h>
2
3 int main(void)
4 {
5     int state;
6
7     FILE * file = fopen("Test.txt", "wt");
8     if(file == NULL) {
9         printf("file open error!\n");
10        return 1;
11    }
12
13    state = fclose(file);
14    if(state != 0) {
15        printf("file close error!\n");
16        return 1;
17    }
18
19    return 0;
20 }
21
```

```
[root@mclab chap7]# vi chap7-1.c
[root@mclab chap7]# gcc -o chap7-1 chap7-1.c
[root@mclab chap7]# ./chap7-1
[root@mclab chap7]# ls
chap7-1  chap7-1.c  Test.txt
[root@mclab chap7]#
```

Test.txt 생성

Formatting Input/Output functions

◆ Formatting functions

	키보드/모니터	선택(키보드/모니터, 파일)
문자 출력	<code>int putchar(int c)</code>	<code>int fputc(int c, FILE* stream)</code>
문자 입력	<code>int getchar(void)</code>	<code>int fgetc(FILE* stream)</code>
문자열 출력	<code>int puts(const char* s)</code>	<code>int fputs(const char* s, FILE* stream)</code>
문자열 입력	<code>char* gets(char* s)</code>	<code>char* fgets(char* s, int n, FILE* stream)</code>
형식 지정 출력	<code>int printf(const* format, ...)</code>	<code>int fprintf(FILE* stream, const char* format, ...)</code>
형식 지정 입력	<code>int scanf(const char* format, ...)</code>	<code>int fscanf(FILE* stream, const char* format, ...)</code>

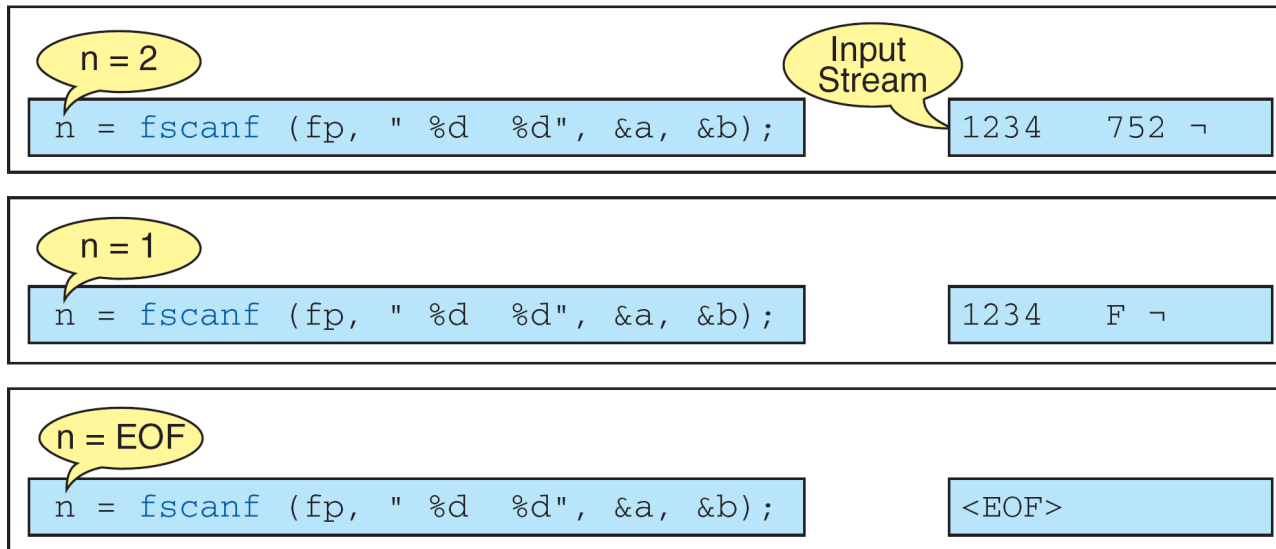
파일 입력 함수

◆ fscanf

- *scanf*와 기능이 거의 같지만 *scanf*는 표준입력(터미널 입력)에 사용되는 반면 *fscanf*는 파일로부터의 입력에 사용된다.

fscanf (fp, “format string”, address list)

- *fscanf*는 리턴 값으로 정상적으로 입력 성공한 변수의 갯수를 반환한다.



파일 출력 함수

◆ fprintf

- *fprintf*는 *printf*와 기능은 거의 같지만 *printf*가 표준 출력(터미널 출력)에 사용되는 반면 *fprintf*는 파일로의 출력에 사용된다.

fprintf (sp, “format string”, value list)

- *ex)*
fprintf (spReport, “\nWelcome to calculator.\n”);
fprintf (spReport, “\nThe answer is %6.2f\n”, x);

Formatting Input/Output functions

◆ fprintf와 fscanf를 사용한 예제

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     FILE *ifp, *ofp;
6     char name[20];
7     int age;
8     double height;
9     int res;
10
11     ifp=fopen("a.txt", "r");
12     if(ifp==NULL) {
13         printf("input file open error!\n");
14         return 1;
15     }
16     ofp=fopen("b.txt", "w");
17     if(ofp==NULL) {
18         printf("output file oepn error!\n");
19         return 1;
20     }
21
22     while (1) { 무한
23         res=fscanf(ifp, "%s%d%lf", name, &age, &height);
24         if(res==EOF) break;
25         fprintf(ofp, "%.1lf %d %s\n", height, age, name);
26     }
27
28     fclose(ifp);
29     fclose(ofp);
30 }
31
```

a.txt

David	25	187.5
Luis	28	173.2
Bill	20	185.4



b.txt

187.5	25	David
173.2	28	Luis
185.4	20	Bill

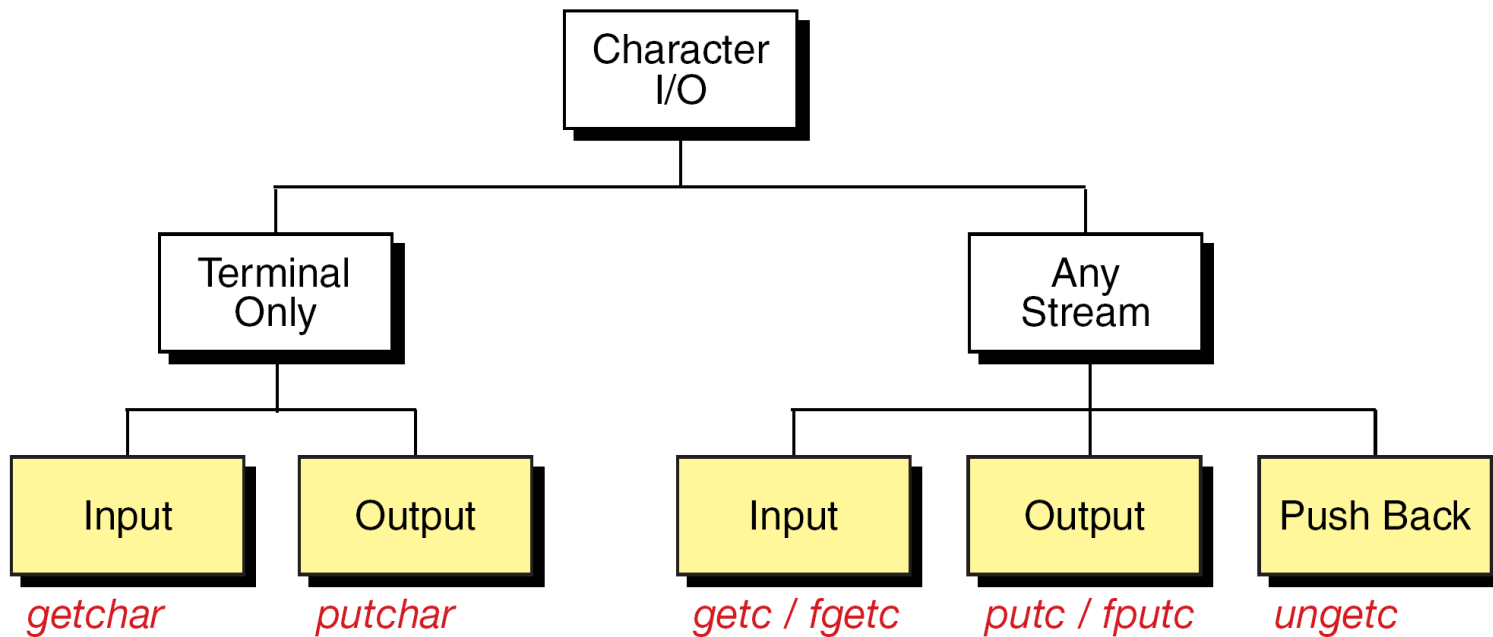
Character Input/Output Functions

◆ Character input functions

- 텍스트 스트림으로부터 한번에 한 글자씩 읽는다.

◆ Character output functions

- 텍스트 스트림에 한번에 한 글자씩 쓴다.



Character Input/Output Functions

◆ Read a Character

- *getchar*: 표준 입력 스트림으로부터 한 글자를 읽고 값을 반환한다.

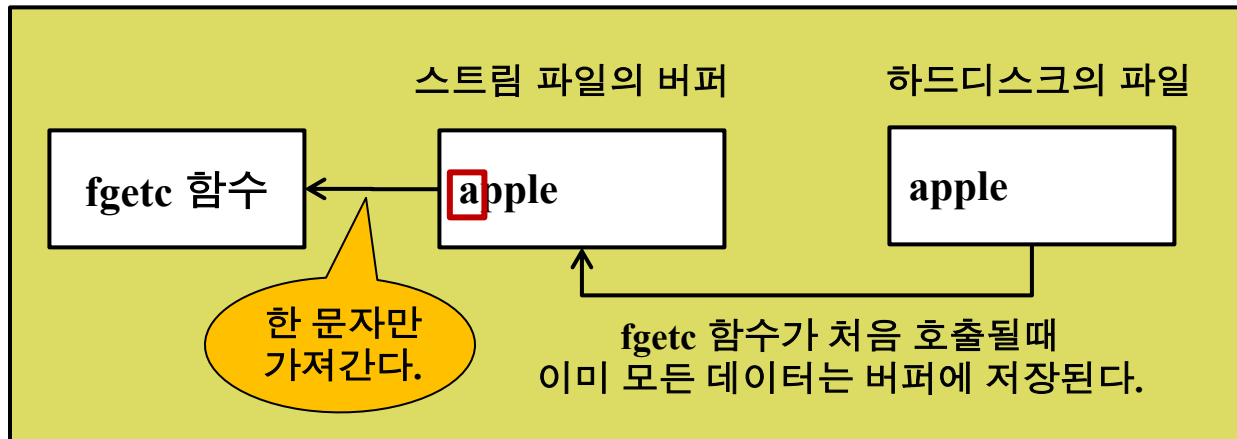
```
int getchar (void);
```

- *getc* and *fgetc*: 파일 스트림으로부터 다음 글자를 읽은 뒤 정수로 변환한다.

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

* int type은 EOF (-1) 때문임. EOF가 아니면 읽은 char를 int형으로 return

ex) nextChar = fgetc (spMyFile);



Character Input/Output Functions

◆ Write a Character

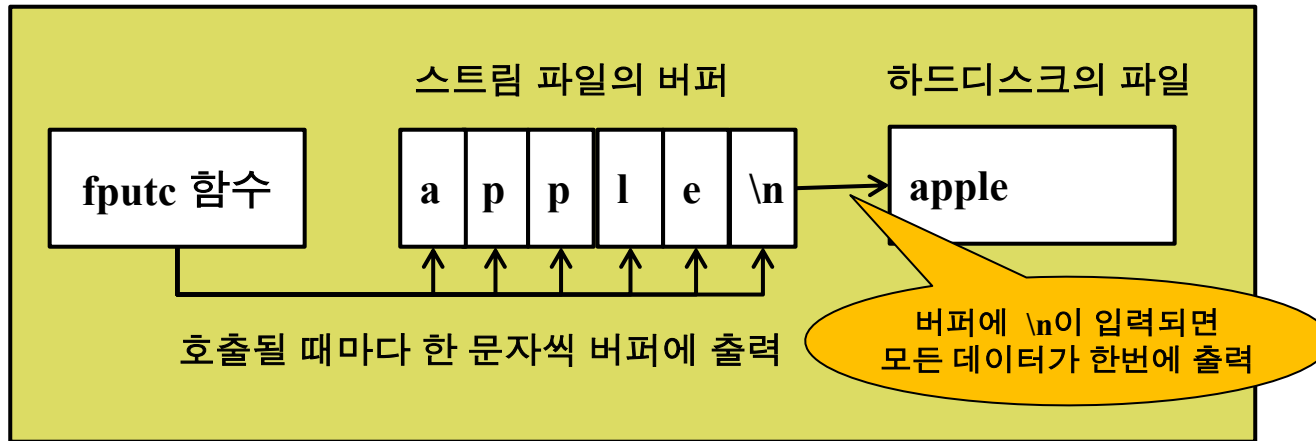
- *putchar*. 모니터에 한 글자를 쓴다.

```
int putchar (int out_char);
```

- *putc* and *fputc*. 파일 스트림에 한 글자를 쓴다.

```
Int fputc (int oneChar, FILE* spOut);
```

ex) *fputc* (oneChar, spMyFile);



- *int* type으로 선언한 이유: 출력이 성공적으로 이루어지면 출력한 char를 *int* 형으로 return.
- 출력이 실패하면 -1 (EOF)가 return 됨

Character Input/Output Functions

◆ fgetc와 fputc를 사용한 예제

```
1 #include <stdio.h>
2
3 int main(void)
4 {
5     char ch;
6
7     while(1) {
8         ch = fgetc(stdin);
9         if(ch==EOF) break;
10        fputc(ch, stdout);
11    }
12    return 0;
13 }
14
```

```
[root@mclab chap7]# vi chap7-3.c
[root@mclab chap7]# gcc -o chap7-3 chap7-3.c
[root@mclab chap7]# ./chap7-3
sogang university
sogang university

[2]+  Stopped                  ./chap7-3
```