Advanced
C Programming & Lab

13. File IO

Sejong University

Outline

1) **File IO**

2) **File IO Procedures**

3) **Text File vs. Binary File**

4) **File IO Function: Text File**

1) File IO

- **Standard IO**

  - Input: a standard input device (keyboard)
    output: a standard output device (monitor)

  - Standard input functions: scanf(), getchar(), gets()

  - Standard output functions: printf(), putchar(), puts()

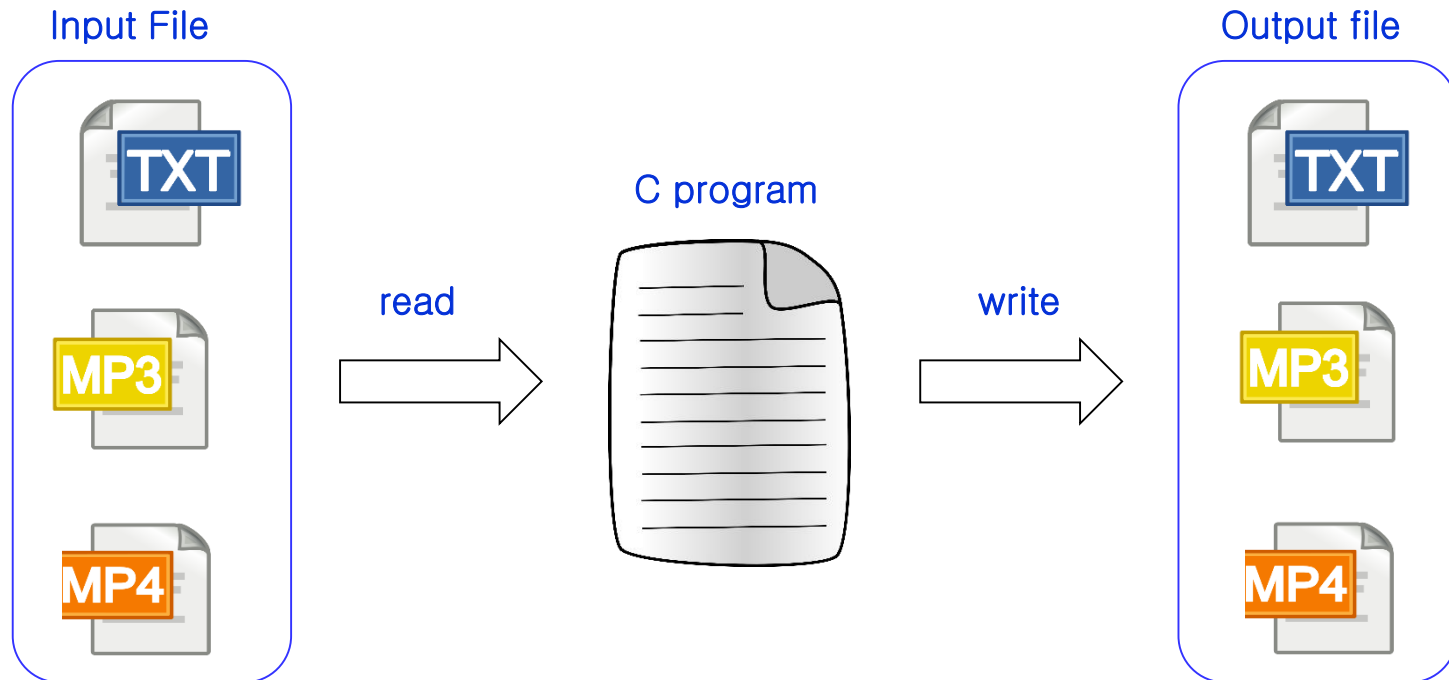 - Terminating a program, input and output results will disappear

C program

Standard
input device

Standard
input functions

Standard
output functions

Standard
output device
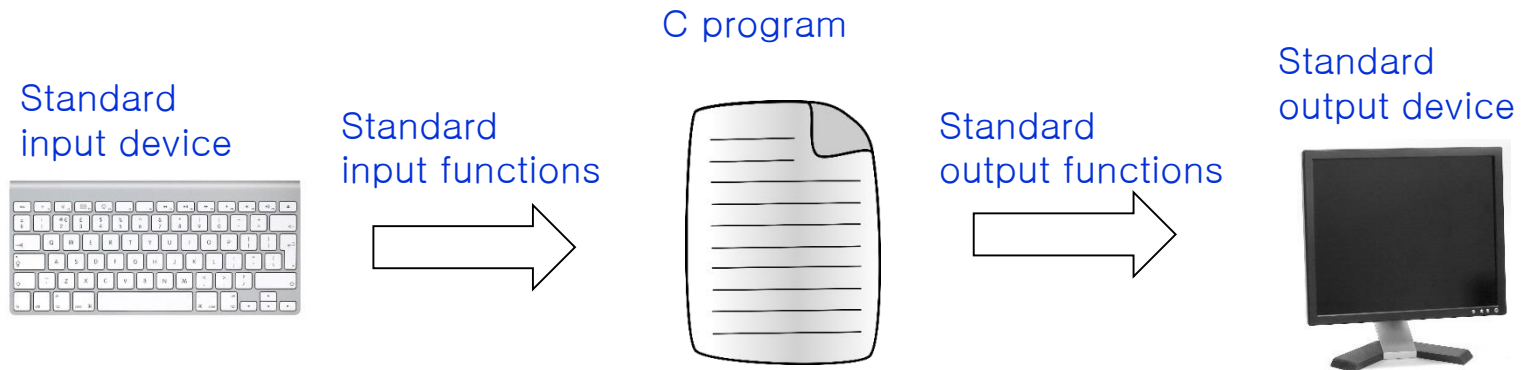
1) File IO

- **Need to store data??**

➜ **File Input/Output**

- ✓ Read data from a file output to a file

- ✓ C provides library functions

Input File

C program

Output file

read

write

1) File IO

- **Data transfer: program - file**

file   stream   C program
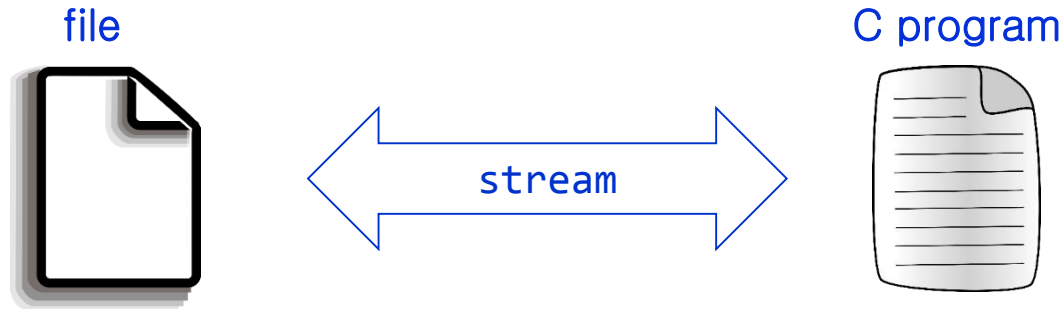
- **Stream**

  - Logical interface between a program and a file (logical data interface)

  - Specifically, use FILE structure and file buffer

  - Consistent I/O → improve efficiency of I/O

    - ✓ Programming, independent of devices

▪ **Execute the following program**

<u>main.c</u>

```c
#include <stdio.h>

int main()
{
        double weight = 78.3;
        int age = 31;

        FILE *fp;
        fp = fopen("test.txt", "w");

        fprintf(fp, "FIRST FILE TEST!\n");
        fprintf(fp, "%.2f %d\n", weight, age);

        fclose(fp);

        return 0;
}
```
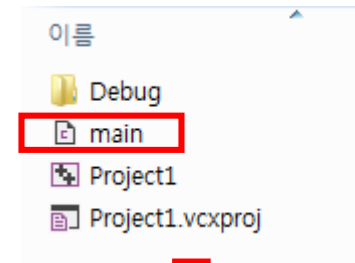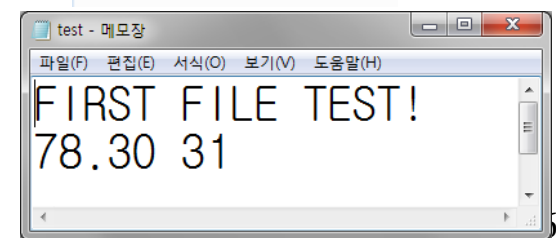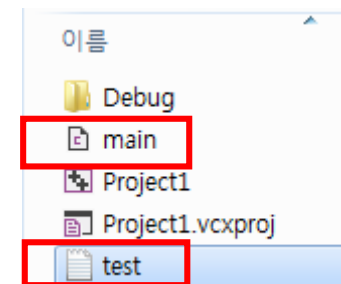
Directory including main.c

Results: create text.txt!

1) File IO

- **Create "test_data.txt" in the current working directory, execute the following program.**

```c
#include <stdio.h>                    main.c
#define SIZE 3
int main()
{

    double weight;
    int age, i;

    FILE *fp;
    fp = fopen("test_data.txt", "r");

    for (i = 0; i < SIZE; i++)
    {
        fscanf(fp, "%lf %d", &weight, &age);
        printf("%.2f %d\n", weight, age);
    }

    fclose(fp);
    return 0;
}
```
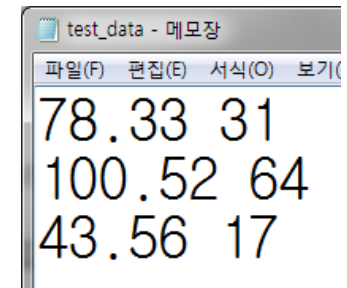
test_data.txt :
(use notepad)

Copy test_data.txt to the directory where main.c is
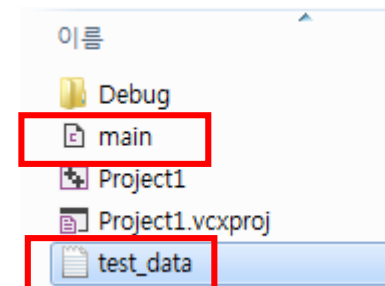
Result
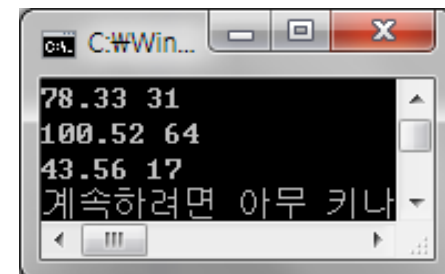
1) File IO

FILE pointer Declaration
- FILE *fp

open file
- fopen()

file read/write
- file input functions:
  fgetc(), fgets(), fscanf(), fread()
- file output functions:
  fputc(), fputs(), fprintf(), fwrite()

close file
- fclose()
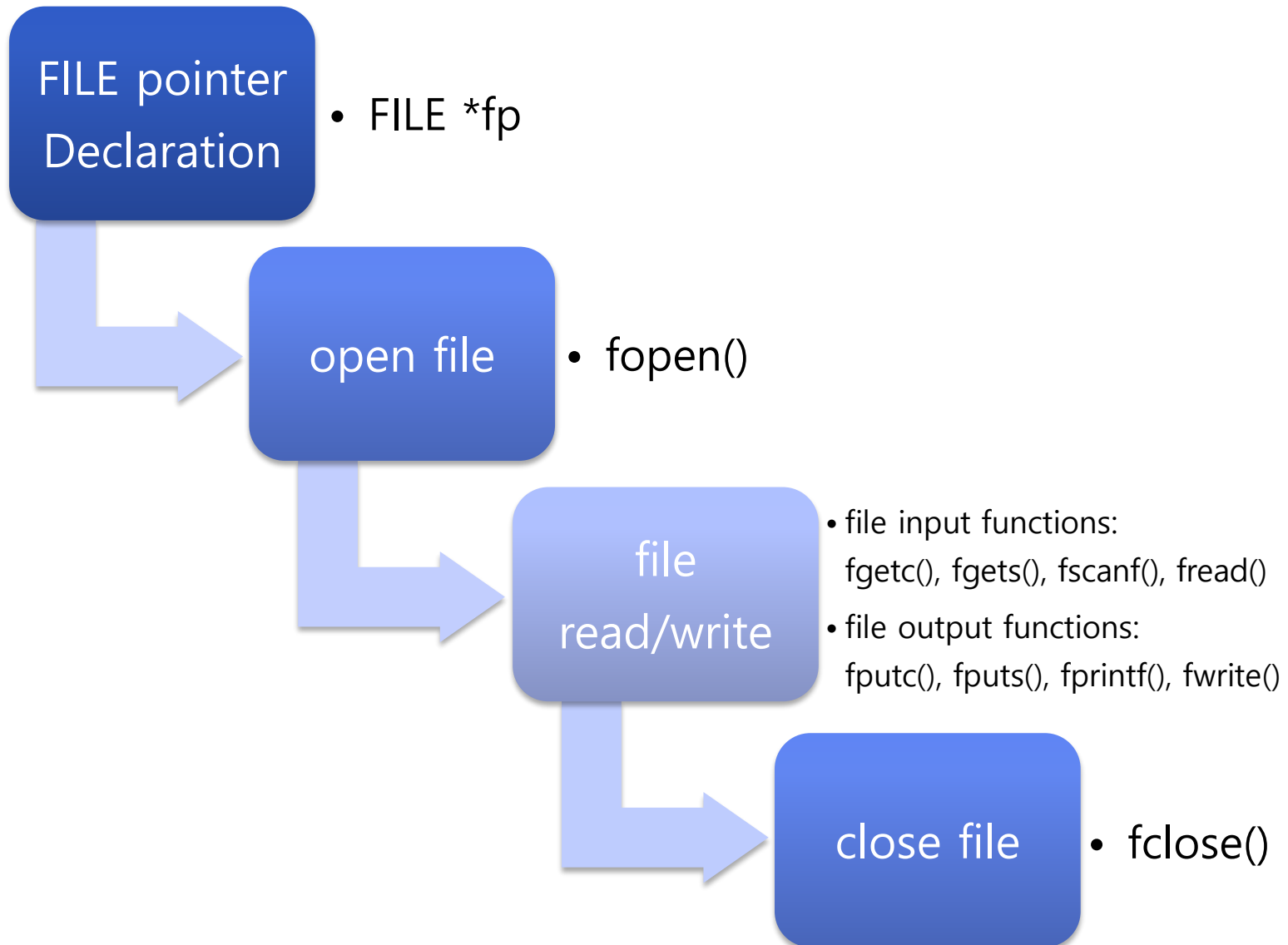
Outline

1) **File IO**

2) **File IO Procedures**

3) **Text File vs. Binary File**

4) **File IO Function: Text File**

2) File IO Procedures

- **Must include <stdio.h>**

- **Declare file pointer**

  - File pointer: pointer to a FILE structure

  - Declare a pointer to a FILE structure

  - Usage

    FILE * file_pointer_name;

    Upper-case!!

2) File IO Procedures

- **file open: fopen() function**

  - Create a IO stream for the given file

  - Return a file pointer to the given file

Function Prototype	**FILE *fopen(char *filename, char *filemode);**	
Function Argument	filename	Name of a file to associate the file steam to
	filemode	Type of stream
Return Type	✓ successful → FILE pointer ✓ failed → NULL	

2) File IO Procedures

- **Function argument: filename (1/3)**

  - The position to find the file to be opened

    - ✓ May depend on environment, settings, and etc

    - ✓ Not specified → **current working directory**

      - ✓ Current working directory = where the current source code is

      - ✓ ex) fopen("test.dat", "filemode");

    filename!

- **Function argument: filename (2/3)**

  - File is not in the current working directory → give **Path** to the file!

    - ✓ Absolute file path

      - ✓ Including root directory, subdirectories

      - ✓ Regardless of computer environment, file path never change

      - ✓ Ex) fopen("C:₩₩C_pro₩₩Project₩₩test.dat", "filemode");

        - » C drive > subdirectory C_pro > subdirectory Project > a file test.dat

        - » ₩₩: ₩ - backslash(₩) twice
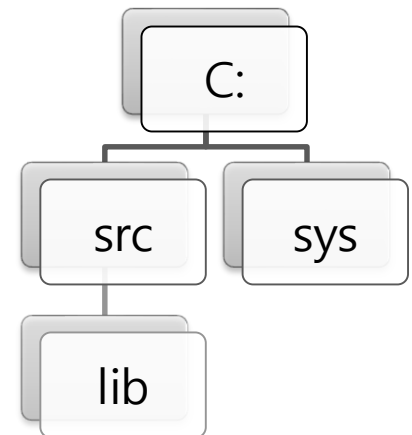
13

2) File IO Procedures

- **Function argument: filename (3/3)**

  - Relative file path

    - ✓ Depend on the computer environment

    - ✓ Based on the current working directory

    - ✓ Ex) current working directory: src

      fopen("lib₩₩data.txt", "filemode");

      fopen("..₩₩sys₩₩data2.txt", "filemode");

< Figure 1 >

2) File IO Procedures

- **Function argument: filemode**

  - Specify the purpose of file opening

  - Mode can prevent mis-usage of a file

< Filemode>

type	mode	meaning	explanation
Input	r	Read	✓ For opening ✓ Cannot open the file → NULL
Output	w	Write	✓ For writing ✓ File does not exist → create a new file ✓ File exists → Delete its content, write the new content
	a	Append	✓ Append to a file ✓ File does not exist → create a new file ✓ File exist → Writhe to the end of the existing file

2) File IO Procedures

- **fopen() function**

FILE *fp;   //FILE structure pointer

fp = fopen("abc.txt", "w");   //abc.txt, open for writing

FILE *fp2;

fp2 = fopen("data/text.dat", "a");

 //subdirectory data, text.dat, open to append to the file

2) File IO Procedures

- **Caution: fopen()**

  - Should check the return type

```
FILE *fp;
fp = fopen("data.txt", "r");
if (fp == NULL)
{
        printf("Couldn't open file!");
        return -1;
}
```

2) File IO Procedures

- **File close: fclose()**

  - Close the file stream to the given file

    - ✓ Writhe the remaining buffered output

Function Prototype	**FILE *fclose(FILE *fp);**	
Function Argument	fp	File pointer to a file to be closed
Return Type	✓ successful → return 0 ✓ failed → return EOF	

※ **EOF (End Of File)**?
✓ Constant (-1) to represent the end of a file
✓ To check whether errors occured or file reading completed

2) File IO Procedures

- **fclose()**

```
FILE *fp;
fp = fopen("test.dat", "r");
if (fp == NULL)
{
          printf("file reading successful!\n");
          return -1;
}
fclose(fp);
printf("file close successful!\n");
```

2) File IO Procedures

- **Practice: file I/O**

  - Open and close a file text1.txt in read mode

  - Open and close a file data1.dat in write mode which is located in the parent directory of the current working directory

  - Open and close a file text2.dat in append mode which is located in the subdirectory Project of the current working directory

2) File IO Procedures

Standard input/output stream: automatic

FILE pointer	Stream	Meaning
stdin	Standard input stream	Input from a keyboard
stdout	Standard output stream	Output to a monitor
stderr	Standard error stream	Error message to a monitor

Outline

3) Text File vs. Binary File

- **Type of file storage**

	Text file	Binary file
properties	✓human-readable characters ✓Easy to read the file ✓Can read the file using notepad ✓All data are converted to character strings ✓Process sequentially	✓Computer-readable data ✓Need a specific application to access the file ✓Cannot read the file using notepad ✓Numerical data are not converted to character strings ✓Take less amount of storage than text files ✓Fast to read and write ✓Store each block of data (Byes) ➜ Random access

3) Text File vs. Binary File

- **fopen() function argument: filemode (1/3)**
  ① Type of file access (refer to p. 16)
  ② Type of file storage

	Text mode	Binary mode
properties	✓ Text file I/O ✓ Automatic newline conversion (OS may differently process it) ✓ Programmer does not need to handle newline conversion	✓ Binary file I/O ✓ Stored as binary ✓ Do not need to mark the end of a line ✓ NULL and newline are treated as data ✓ Better to store numerical data

3) Text File vs. Binary File

- **fopen() function argument: filemode (2/3)**

Text	Binary	Explanation
r (rt)	rb	✓ Open for reading ✓ Cannot open a file → NULL
w (wt)	wb	✓ Open for writing ✓ File does not exist → Create a new file ✓ File exists → Delete its content, write the new content
a (at)	ab	✓ Open to append to the file ✓ File does not exist → Create a new file ✓ File exists → Writhe to the end of the existing file

3) Text File vs. Binary File

- **fopen() function argument: filemode (3/3)**

Text	Binary	explanation
r+	rb+	✓ Open for reading and writing ✓ File should exist
w+	wb+	✓ Open for reading and writing ✓ File does not exist → Create a new file ✓ File exists → Delete its content, write the new content
a+	ab+	✓ Open to read and append to the file ✓ File does not exist → Create a new file ✓ File exists → Can read from a random position, but write to the end of the file

3) Text File vs. Binary File

- **Text mode: fopen()**

FILE *fp;   //FILE structure pointer

fp = fopen("test.txt", "r");   //test.txt, read mode

- **Binary mode: fopen()**

FILE *fp;   //FILE structure pointer

fp = fopen("test.dat", "rb");   //test.dat, binary read mode

3) Text File vs. Binary File

- **File I/O functions**

Type	Unit	Input	Output
Text File	Character	fgetc()	fputc()
	Strings	fgets()	fputs()
	Specified Format	fscanf()	fprintf()
Binary File	Block	fread()	fwrite()

※ C provides file I/O functions in stdio.h

3) Text File vs. Binary File

- **File I/O functions**

  - fopen() function argument for file input

    - ✓ filename: file to read

    - ✓ filemode (read mode)

      - ✓ Text file: "r"

      - ✓ Binary file: "rb"

  - fopen() function argument for file output

    - ✓ filename: file to write

    - ✓ filemode (write or append mode)

      - ✓ Text file: "w" or "a"

      - ✓ Binary file: "wb" or "ab"

Outline

4) File IO Function: Text File

▪ **Formatted input: fscanf()**

- Read formatted input from a stream

- Can read several types of data (integer, character, strings, etc)

- First argument is a FILE pointer, the rest is the same with scanf()

Function Prototype	**int *fscanf(FILE *fp, char *format, arg1, arg2, …);**	
Function Argument	fp	FILE pointer
	Format	Format specifiers
	arg1, arg2, …	List of variables
Return Type	✓ Successful → Number of inputs ✓ End-of-File/error → EOF	

4) File IO Function: Text File

- **fscanf() function**

```
char str[10];
int num;
FILE *fp = fopen("data.txt", "r");
if (fp == NULL)
{
        printf("Couldn't open file!");
        return -1;
}
fscanf(fp, "%s %d", str, &num);
```

➔ Read strings and integers using fp associated with data.txt,

Store them in an array str and integer num

4) File IO Function: Text File

- **Formatted output: fprintf()**

  - Formatted output to a stream

  - First argument is a FILE pointer, the rest is the same with printf()

Function Prototype	**int *fprintf(FILE *fp, char *format, arg1, arg2, ...);**	
Function Argument	fp	FILE pointer
	Format	Format specifiers
	arg1, arg2, ...	List of variables
Return Type	✓ successful → number of inputs (Byes) ✓ failed/error → negative number	

4) File IO Function: Text File

- **fprintf() function**

```
int age = 25;
FILE *fp = fopen("data.txt", "w");
if (fp == NULL)
{
        printf("Couldn't open file!");
        return -1;
}
fprintf(fp, "Age: %d", age);
fprintf(stdout, "Age: %d", age);     // printf("Age: %d", age);
```

➔ Write "Age: 25" in a file data.txt using fp, same as on a monitor

4) File IO Function: Text File

- **Get a character: fgetc()**
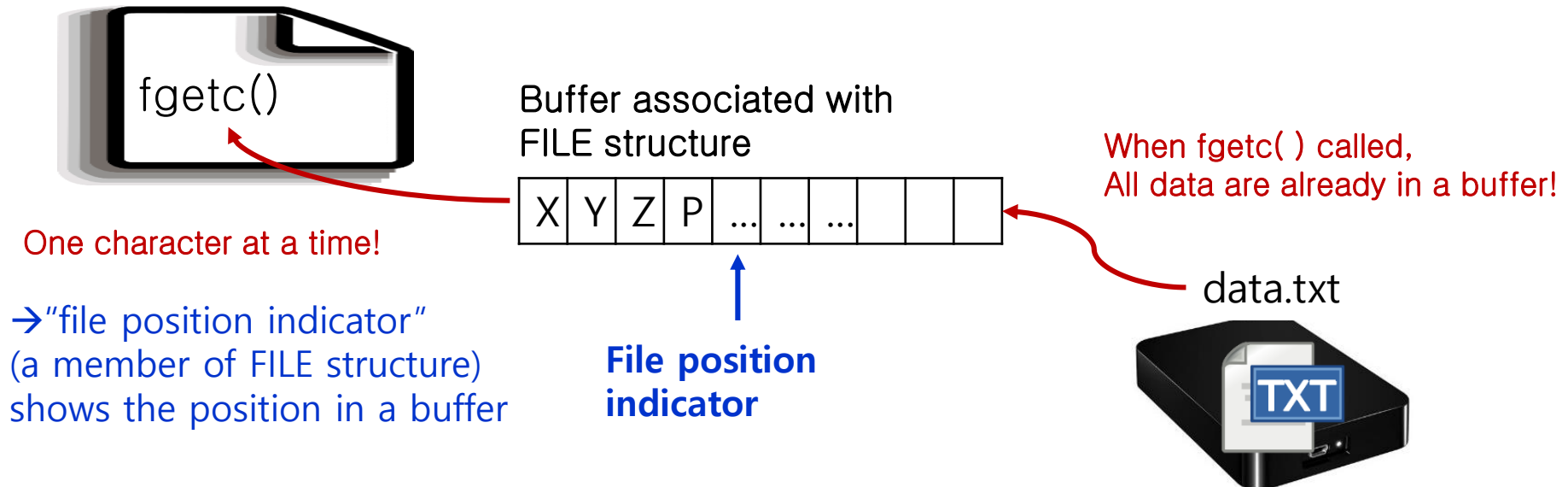 - Get a character from a stream

Function Prototype	**int fgetc(FILE *fp);**	
Function Argument	fp	FILE pointer
Return Type	✓ successful → received character ✓ Failed/error → EOF	

- **Write a character: fputc()**
  - Write a character to a stream

Function Prototype	**int fputc(int char, FILE *fp);**	
Function Argument	char	Character to write
	fp	FILE pointer
Return Type	✓ Successful → written character ✓ Failed/error → EOF	

4) File IO Function: Text File

- **fgetc() procedures**

fgetc()

One character at a time!

→"file position indicator"
(a member of FILE structure)
shows the position in a buffer

Buffer associated with
FILE structure

| X | Y | Z | P | ... | ... | ... | | | |

**File position
indicator**

When fgetc( ) called,
All data are already in a buffer!

data.txt

TXT

- **fputs() procedures**
  - Use a buffer just like fgetc()
  - Write to a buffer when receiving a character
    Write to a disk when receiving a new line

4) File IO Function: Text File

- **File I/O**

```c
#include <stdio.h>

int main()
{
    FILE *fp1, *fp2;
    char ch;

    fp1 = fopen("input.txt", "r");
    if (fp1 == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
    fp2 = fopen("output.txt", "w");
    if (fp2 == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
```

```c
    while((ch = fgetc(fp1)) != EOF)
    {
        printf("%c", ch);
        fputc(ch, fp2);
    }

    fclose(fp1);
    fclose(fp2);

    return 0;
}
```

▪ **Get characters from a stream: fgets()**

- Read characters/strings from a file

- **New lines** in a file are treated as **characters/strings**

- Maximum number of characters to be read is determined

  ✓ (Maximum number of characters – 1) + NULL

- Happened to have a new line?

➔ Characters before the new line are returned

4) File IO Function: Text File

- **Get characters from a stream: fgets()**

Function Prototype	char *fgets(char *s, int n, FILE *fp);	
Function Argument	s	Pointer to a string
	n	Maximum number of characters
	fp	FILE pointer
Return Type	✓ Successful → string s ✓ Failed/error → NULL	

4) File IO Function: Text File

- **Example: fgets()**

info.txt

  char str1[20], str2[20], str3[20];
  FILE *fp = fopen("info.txt", "r");

Neungdong-ro,↵
Gwangjin-gu, Seoul, Korea.↵

1) fgets(str1, 20, fp);

| N | e | u | n | g | d | o | n | g | - | r | o | , | ₩n | ₩0 | | | | | |

→ Characters before the new line are written to an array str1 (including NULL)

2) fgets(str2, 20, fp);

| G | w | a | n | g | j | i | n | - | g | u | , | | S | e | o | u | l | ₩0 |

→ 19 characters are written to an array str2 (including NULL)

3) fgets(str3, 20, fp);

| , | | K | o | r | e | a | . | ₩n | ₩0 | | | | | | | | | | |

→ Characters before the new line are written to an array str3 (including NULL)

4) File IO Function: Text File

- **Write strings to a stream: fputs()**

  - Write characters/strings to a stream

  - Omit NULL and the new line

Function Prototype	**int fputs(char *str, FILE *fp);**	
Function Argument	str	Characters to be written
	fp	FILE pointer
Return Type	✓ Successful → Number of bytes to be written ✓ Failed/error → EOF	

4) File IO Function: Text File

- **File I/O**

```
#include <stdio.h>

int main()
{
    char str[100];
    FILE *fp1, *fp2;

    fp1 = fopen("input.txt", "r");
    if (fp1 == NULL)
    {
       printf("Couldn't open file!");
       return -1;
    }
    fp2 = fopen("output.txt", "w");
    if (fp2 == NULL)
    {
        printf("Couldn't open file!");
        return -1;
     }
```

```
while(fgets(str, sizeof(str), fp1) != NULL)
{
        printf("%s", str);
        fputs(str, fp2);
 }

  fclose(fp1);
  fclose(fp2);

  return 0;
}
```

4) File IO Function: Text File

- **Check "End-Of-File "**

  ① Return Type

function	
fgetc()	At the end of a file, return EOF(-1)
fgets()	At the end of a file, return NULL(0)
fscanf()	At the end of a file, return EOF(-1)

  ② Use feof()

4) File IO Function: Text File

- **feof()**

  - Check whether the end-of-file indicator associated with the stream is set

  - Must include <stdio.h>

Function Prototype	**int feof(FILE *fp);**	
Function Argument	fp	FILE pointer
Return Type	✓ End-of-file → return a non-zero value ✓ no → return 0	

- **EOF vs. feof()**

  - At the end of any file, EOF exists → EOF is a part of a file

  - feof() returns 0 when it reaches EOF!
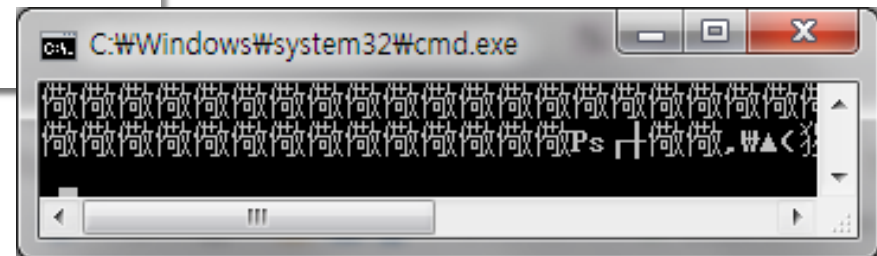
4) File IO Function: Text File

- **Caution: feof()**
  - Ex1) Empty the file data.txt, and execute the following?

```
while(!feof(fp))
    {
        fgets(str, sizeof(str), fp);
        printf("%s", str);
    }
    fclose(fp);
    return 0;
}
```

```
#include <stdio.h>

int main()
{
    FILE *fp;
    char str[100];

    fp = fopen("data.txt", "r");
    if (fp == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
```
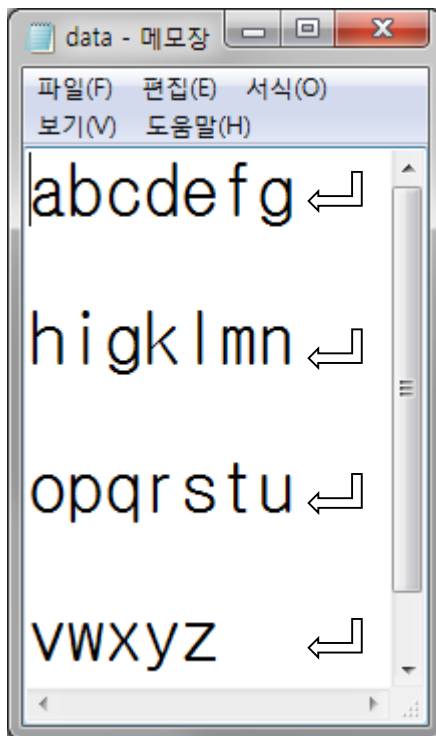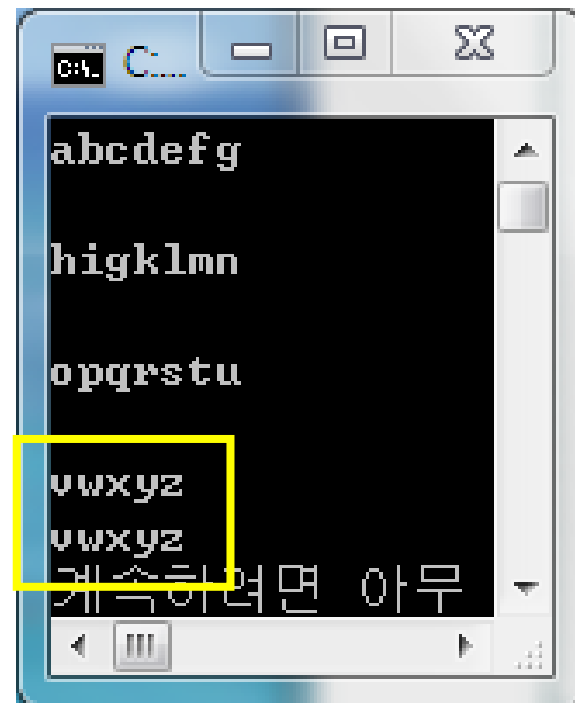
4) File IO Function: Text File

- **Caution: feof()**
  - Ex2) Create a file data.txt as below, Execute the previous program?

<data.txt>

<Result>

4) File IO Function: Text File

- **Caution: feof()**

  - Why?

    - ✓ Automatically add the special character (^Z), indicating the end of a file

      - ✓ Empty file contains ^Z (Ex1)

      - ✓ Issues with the position of ^Z (Ex2)

    - ✓ feof(fp) return value = 0 (False)

      - ✓ Passing ^Z, feof(fp) return a non-zero value (True)

  - Solution

    - ✓ Read data first, then check if it reaches the end-of-file

4) File IO Function: Text File
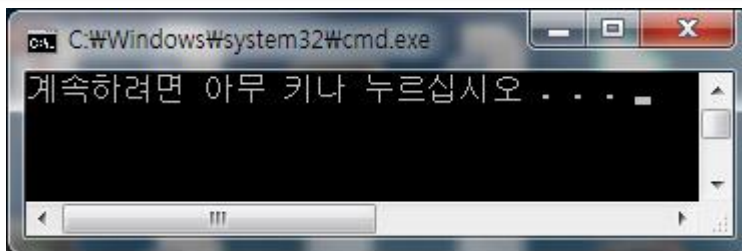
- **Caution: feof()**
  - Modified code

```c
#include <stdio.h>

int main()
{
    FILE *fp;
    char str[100];

    fp = fopen("data.txt", "r");
    if (fp == NULL)
    {
        printf("Couldn't open file!");
        return -1;
    }
```

```c
    fgets(str, sizeof(str), fp);
    while(!feof(fp))
        {
                printf("%s", str);
                fgets(str, sizeof(str), fp);
        }
    fclose(fp);
    return 0;
}
```

<Ex1>

<Ex2>