Ch.15 File Input/Output

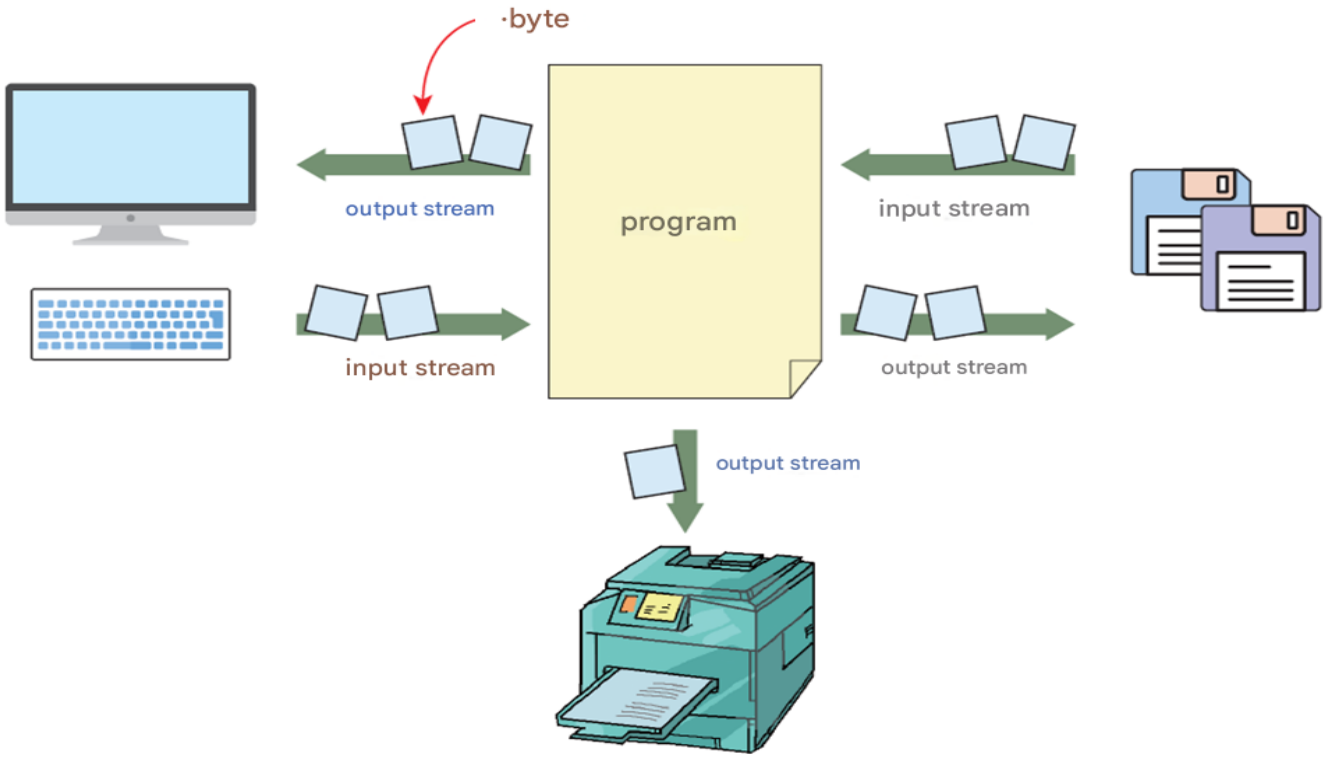
What you will learn in this chapter

- The concept of strips
- Standard Input/Output
- File Input/Output
- Input/output related functions

Learn about concepts and functions related to input/output .

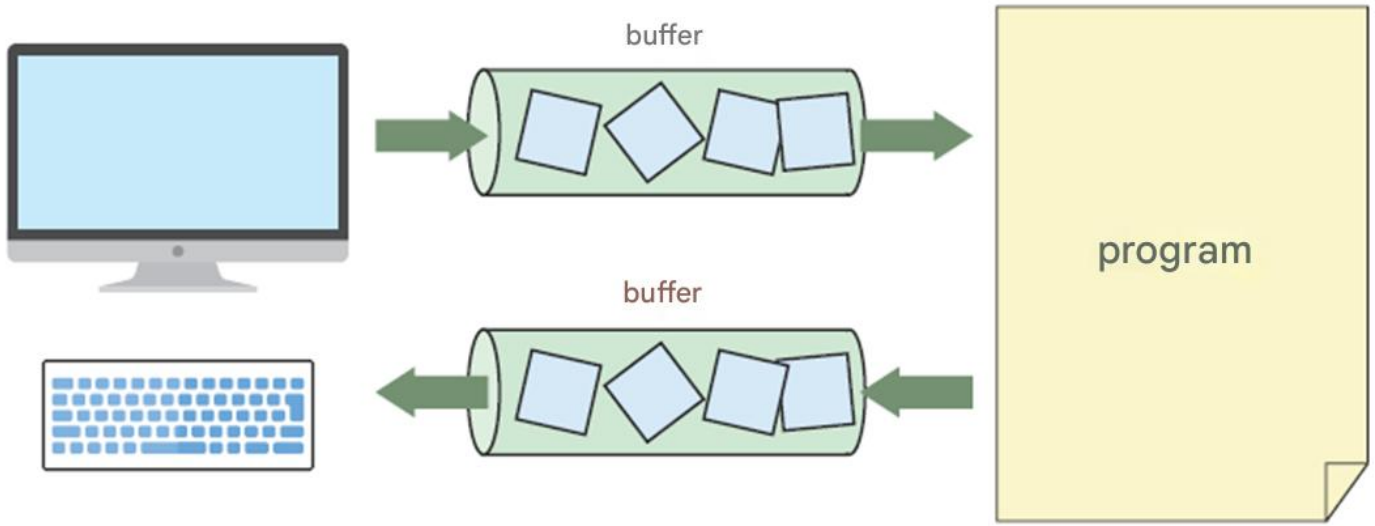
The concept of stream

- Stream : Thinking of input and output as a flow of bytes.
  Data path



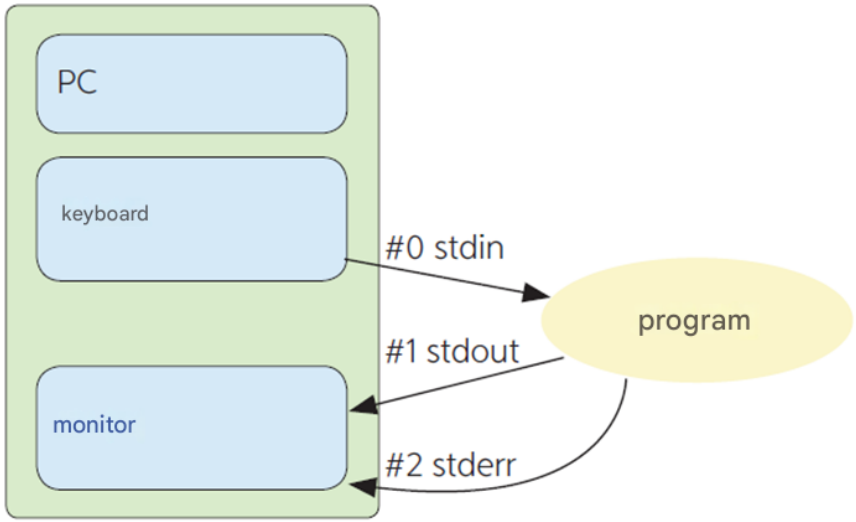
Streams and Buffers

- A stream contains a buffer by default in C.



Standard Input/Output Streams

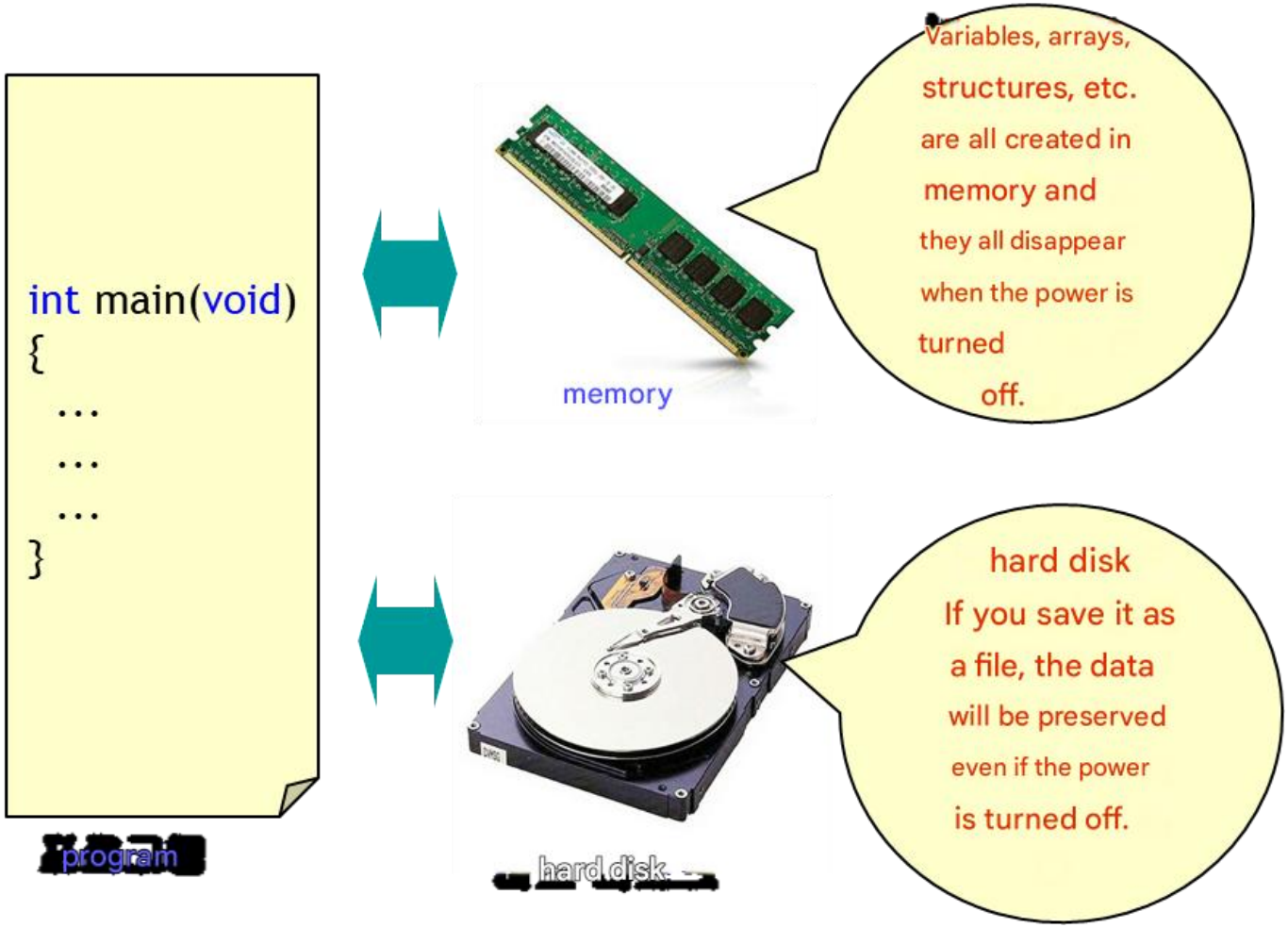
name	stream	connection device
stdin	standard input stream	keyboard
stdout	standard output stream	monitor screen
stderr	standard error stream	monitor screen



Classification of input/output functions

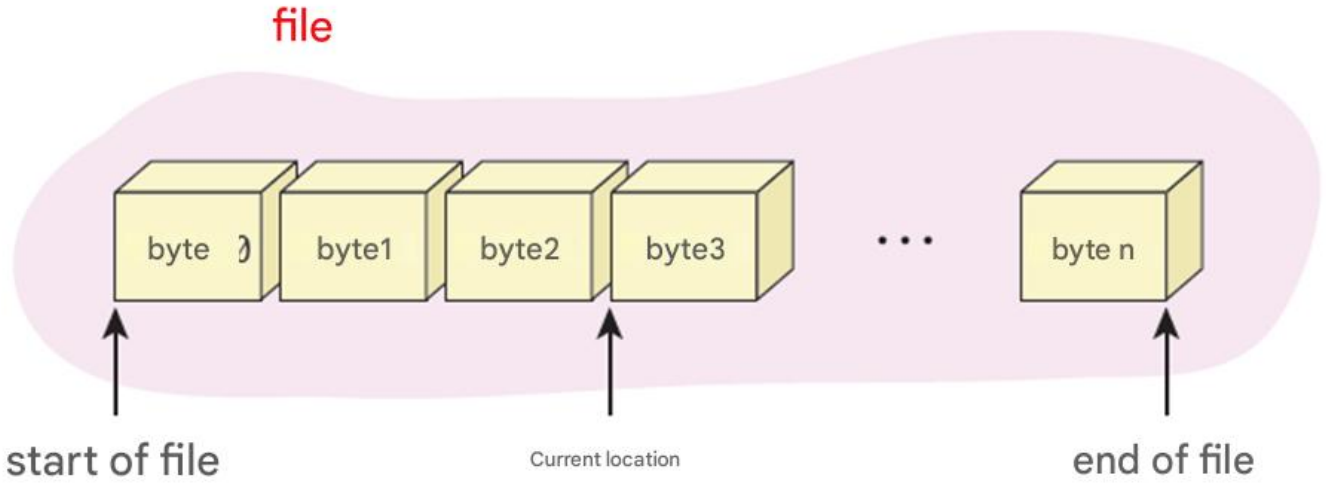
form \ stream	standard stream	normal stream	explanation
Unformatted input/output (character format)	getchar()	fgetc(FILE *f,...)	Character input function
	putchar()	fputc(FILE *f,...)	Character output function
	gets_s()	fgets(FILE *f,...)	string input function
	puts()	fputs(FILE *f,...)	string output function
Formatted input/output (integer, real number etc)	printf()	fprintf(FILE *f,...)	Formatted output function
	scanf()	fscanf(FILE *f,...)	Formatted input function

Why do I need files?



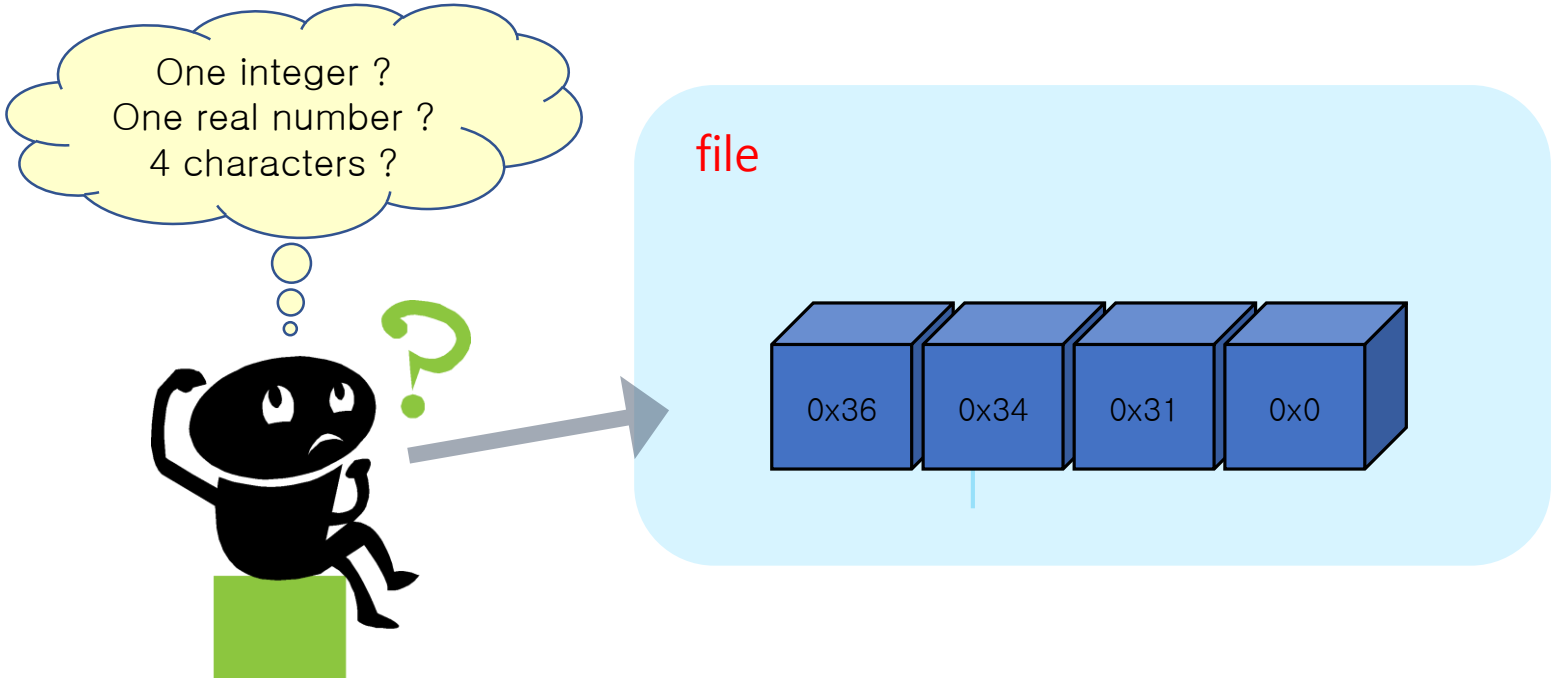
The concept of a file

- A file in C is a series of contiguous bytes.
- All file data is eventually converted to bytes and stored in a file.
- It is entirely up to the programmer to interpret these bytes.



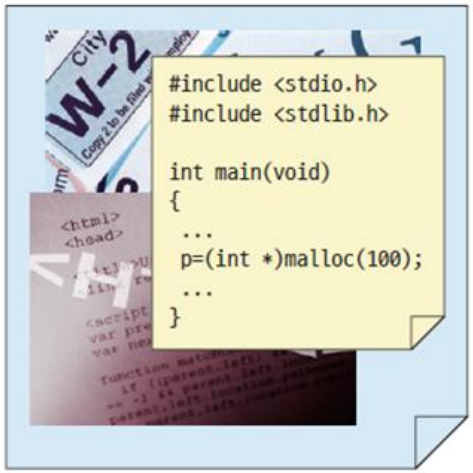
file

- If a file contains 4 bytes, it can be interpreted as either integer data of type int or real number data of type float.

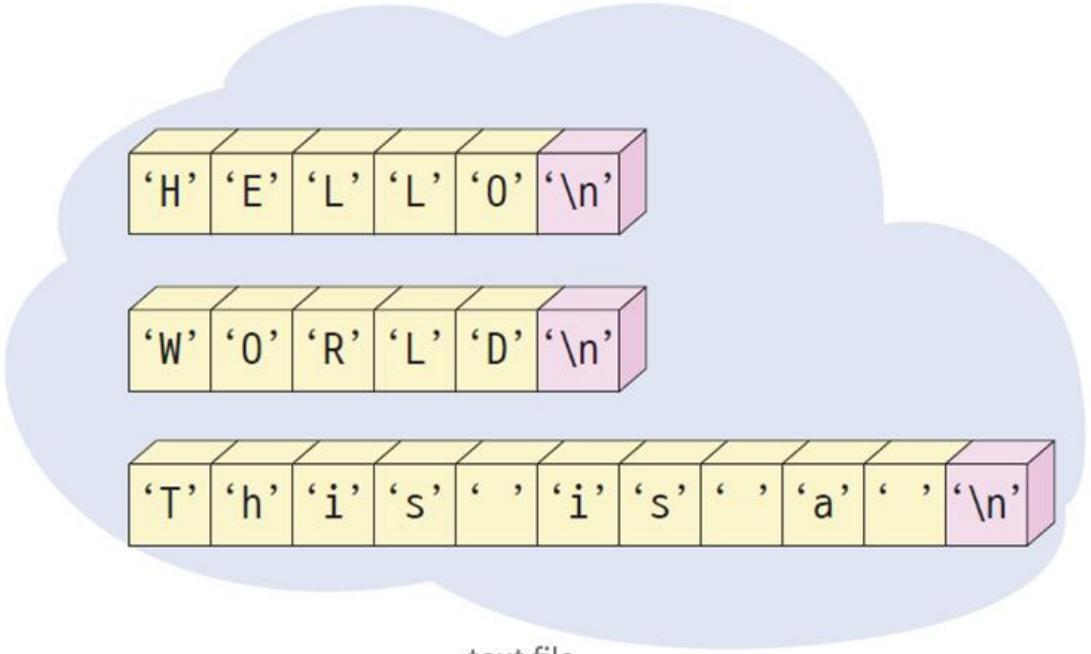


text file

- A text file is a file that contains human-readable text.
  - (Example) C program source file or notepad file
- Text files are saved using ASCII codes.
- A text file is made up of a sequence of lines.



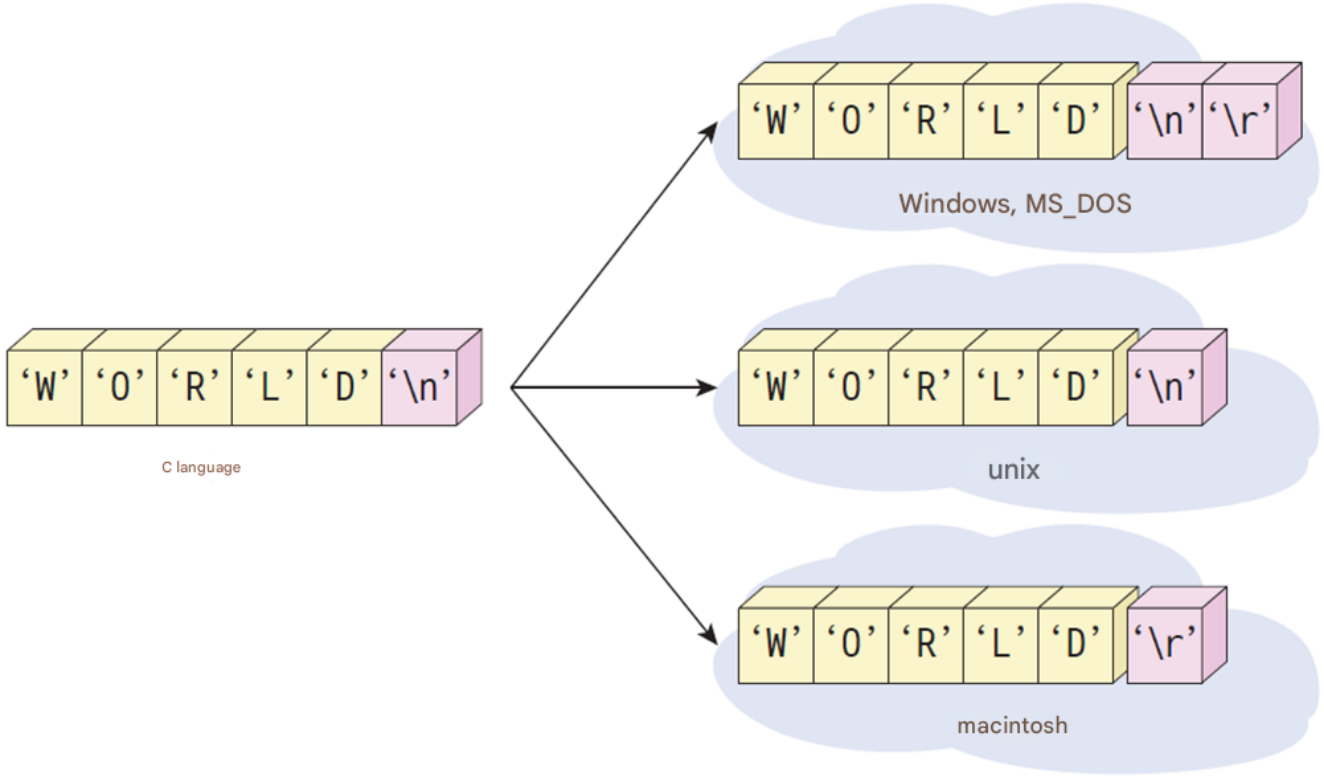
Text file: a file consisting of characters



text file

text file

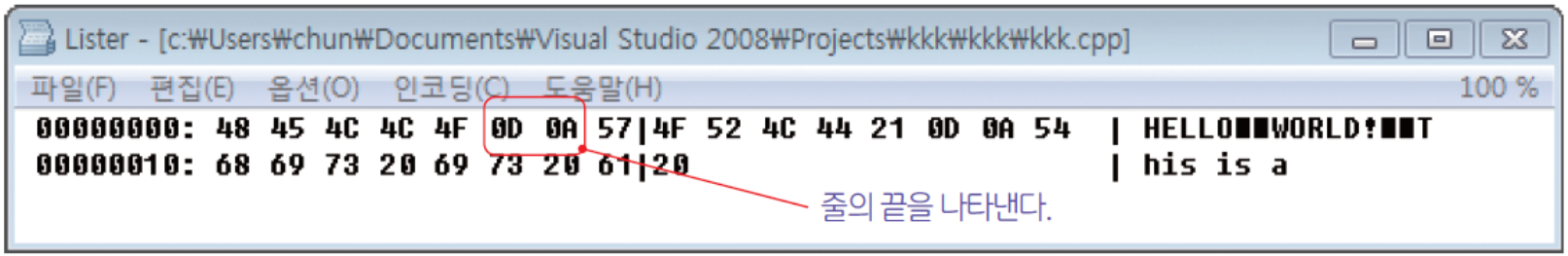
- Each operating system has a different way of displaying line breaks.



Text file in windows

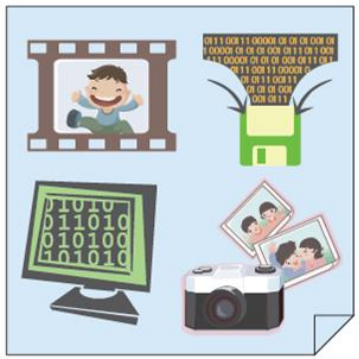
Notepad

- For example, in Windows, text files are saved as shown in Figure 15-7 .

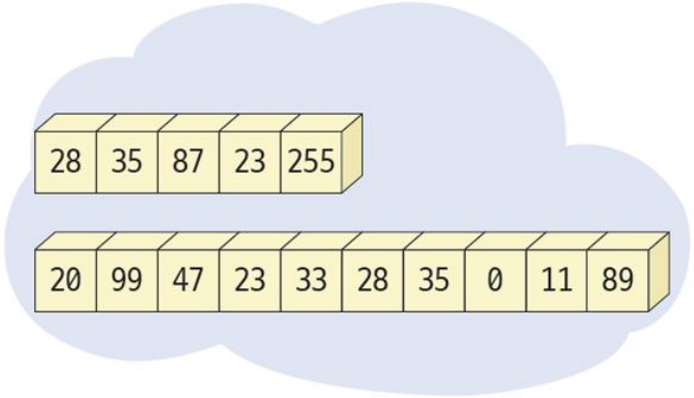


binary file

- Binary files are files that cannot be read by humans but can be read by computers.
- A file that directly stores binary data.
- Binary files, unlike text files, are not separated into lines.
- All data is input/output without being converted to strings.
- Binary files can only be read by certain programs.
- (Example) C program executable file, sound file, image file



Binary file: A file consisting of data.



binary file

binary (image & sound file)

File Type	Header Element	Description
JPG	SOI (Start of Image)	File always begins with bytes `FF D8`.
	APP0 / JFIF Segment	Usually starts with `FF E0` followed by `"JFIF"`; identifies the file format.
	EXIF Metadata**	Optional block containing camera info, orientation, timestamps, etc.
	SOF (Start of Frame)	Contains image width, height, and color component details.
	SOS (Start of Scan)	Marks the beginning of compressed image data.
MP3	Frame Sync	Begins with binary `11111111111` (hex `FF F?`); identifies MP3 frame start.
	MPEG Version	Indicates MPEG-1, MPEG-2, or MPEG-2.5 audio format.
	Layer	Usually Layer III (MP3).
	Bitrate Field	Encodes the audio bitrate (e.g., 128 kbps).
	Sampling Rate Field	Stores sampling frequency (e.g., 44.1 kHz).
	ID3 Tag (Optional)	Starts with `"ID3"`; contains metadata like title, artist, album.

Overview of file processing

- When handling files, you must follow this order :

- Disk files are accessed using the FILE structure.
- A file pointer is a pointer to a FILE structure.

Open file

Syntax open file

yes

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

file name

file mode

FILE structure

- fopen () creates a file with the given file name and returns a FILE pointer .

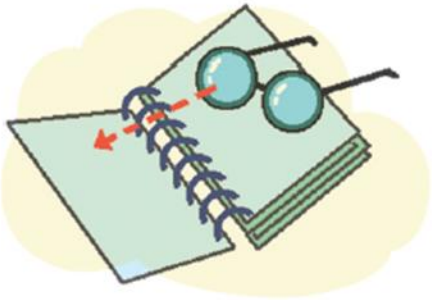
```
struct _ iobuf {
        char * _ptr ;
        int _ cnt ;
        char *_base;
        int _flag;
        int _file;
        int _ charbuf ;
        int _ bufsiz ;
        char * _tmpfname ;
};
typedef struct _ iobuf FILE;
```

```
typedef struct _IO_FILE {
    int _flags;               // File status flags
    char* _IO_read_ptr;       // Current read pointer
    char* _IO_read_end;       // End of read buffer
    char* _IO_read_base;      // Start of read buffer
    char* _IO_write_base;     // Start of write buffer
    char* _IO_write_ptr;      // Current write pointer
    char* _IO_write_end;      // End of write buffer
    char* _IO_buf_base;       // Buffer start
    char* _IO_buf_end;        // Buffer end
    char* _IO_save_base;      // Saved position for backup
    char* _IO_backup_base;    // Backup base
    char* _IO_save_end;       // Saved end pointer
    struct _IO_marker* _markers;  // For tracking positions (ungetc, etc.)
    struct _IO_FILE* _chain;      // Linked list of open FILEs
    int _fileno;              // File descriptor
    int _flags2;              // Additional flags
    long _old_offset;         // Old file offset
    unsigned short _cur_column;  // Column number for wide chars
    signed char _vtable_offset;  // Virtual table offset
    char _shortbuf[1];        // Short buffer
    void* _lock;              // Lock for thread safety
    long _offset;             // Current offset
    // ... (more internal fields, platform-dependent)
} FILE;
```

File Mode

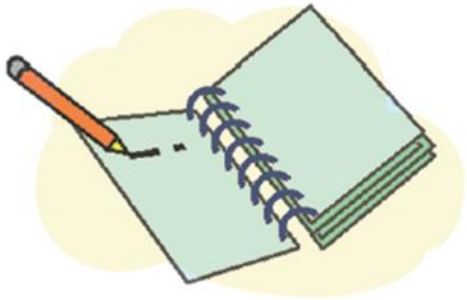
mode	explanation
"r"	Opens a file in read mode. If the file does not exist, an error occurs.
"w"	Creates a new file in write mode. If the file already exists, its contents are erased.
"a"	Opens a file in append mode. If the file already exists, the data is appended to the end of the file. If the file does not exist, a new file is created.
"r+"	Opens a file in read mode. Can be switched to write mode. The file must exist.
"w+"	Creates a new file in write mode. Can be switched to read mode. If the file already exists, its contents will be erased.
"a+"	Opens a file in append mode. Can switch to read mode. Appending data moves the EOF marker to the end of the appended data. If the file does not exist, a new file is created.
"t"	Opens the file in text file mode.
"b"	Opens a file in binary file mode.

Basic file mode



"r"

Read from the beginning of the file



"w"

Write from the beginning of the file.
If the file exists, its existing contents
will be erased

"a"

Write from the beginning of the file.
If the file does not exists,
It is created.

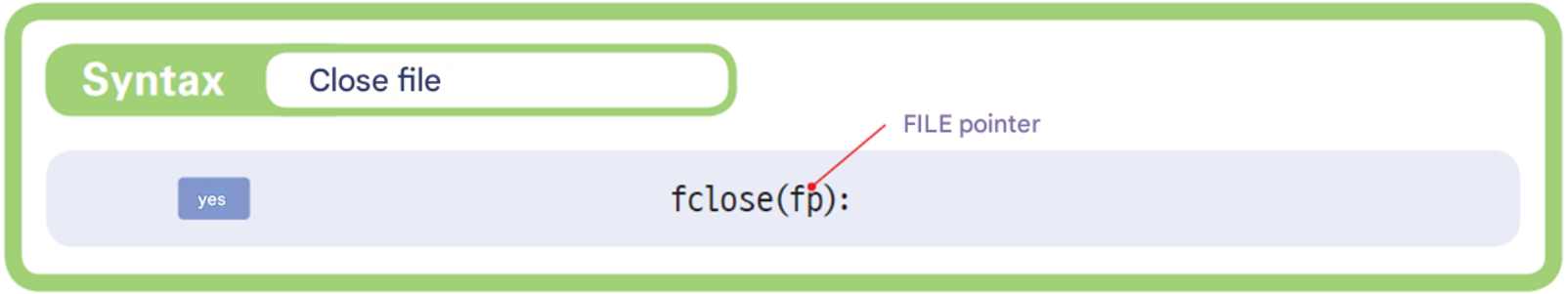
Things to note

- You can append "t" or "b" to the basic file mode .

- "a" or "a+" mode is called **append mode.** When a file is opened in append mode, all write operations occur at the end of the file. Therefore, any existing data in the file is never erased.

- the "r+", "w+", or "a+" file mode is specified, both reading and writing are possible. This mode is called **update mode .**
  To switch from read mode to write mode, or from write mode to read mode, you must call one of fflush(), fsetpos(), fseek(), or rewind() .

Close file



Example

```
#include < stdio.h >
int main( void )
{
        FILE * fp = NULL;

        fp = fopen ( "sample.txt" , "w" );

        if ( fp == NULL )
            printf ( " file opening Failed \n" );
        else
            printf ( " file opening Success \n" );

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

File opening success

File deletion example

```
#include < stdio.h >

int main( void )
{
    if (remove( "sample.txt" ) == -1)
        printf ( "sample.txt cannot be deleted .\n" );
    else
        printf ( "sample.txt has been deleted .\n" );
    return 0;
}
```

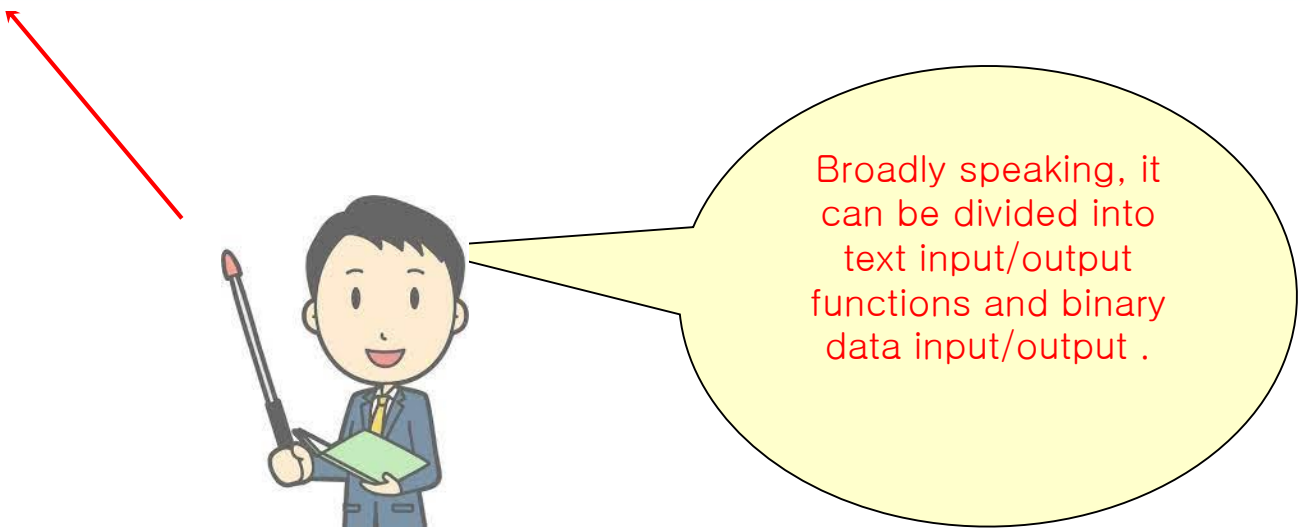
```
sample.txt has been deleted .
```

Other useful functions

function	explanation
`int foef(FILE *stream)`	Returns true when the end of the file is reached.
`int rename(const char *oldname, const char *newname)`	Change the name of the file.
`FILE *tmpfile()`	Creates and returns a temporary file.
`int ferror(FILE *stream)`	Returns the error status of the stream. If an error occurs, true is returned.

File input/output functions

type	input function	output function
Text Data IO function	`int fgetc(FILE *fp)`	`int fputc(int c, FILE *fp)`
Text Data IO function	`char *fgets(char *buf, int n, FILE *fp)`	`int fputs(const char *buf, FILE *fp)`
Text Data IO function	`int fscanf(FILE *fp, ...)`	`int fprintf(FILE *fp,...)`
Binary data IO function	`size_t fread(char *buffer, int size, int count, FILE *fp)`	`size_t fwrite(char *buffer, int size, int count, FILE *fp)`



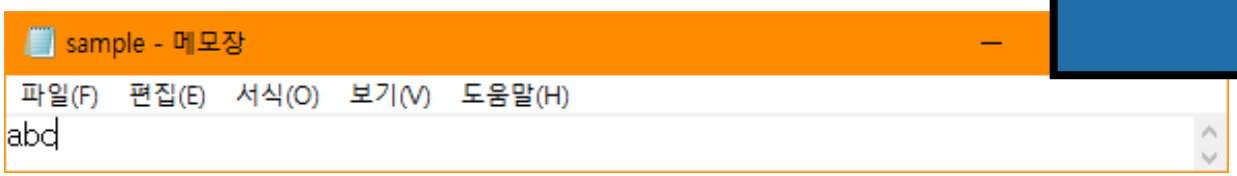
Character unit input/output

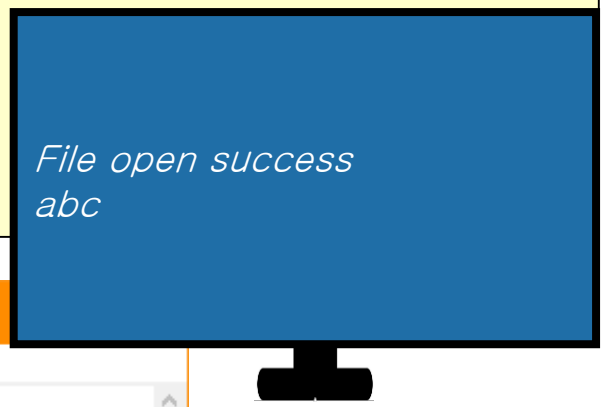
```c
#include < stdio.h >
int main( void )
{
        FILE * fp = NULL;

        fp = fopen ( "sample.txt" , "w" );
        if ( fp == NULL )
            printf ( " file opening Failed \n" );
        else
            printf ( " file opening Success \n" );

        fputc ('a', fp );
        fputc ('b', fp );
        fputc ('c', fp );
        fclose ( fp );
        return 0;
}
```

Character unit input/output

```
#include < stdio.h >
int main( void )
{
        FILE * fp = NULL;
        int c;
        fp = fopen ( "sample.txt" , "r" );
        if ( fp == NULL )
                printf ( " file opening Failed \n");
        else
                printf ( " file opening Success \n");

        while ((c = fgetc ( fp )) != EOF )
                 putchar (c);
        fclose ( fp );
        return 0;
}
```

must be declared as an integer variable. The reason is explained in the next slide .

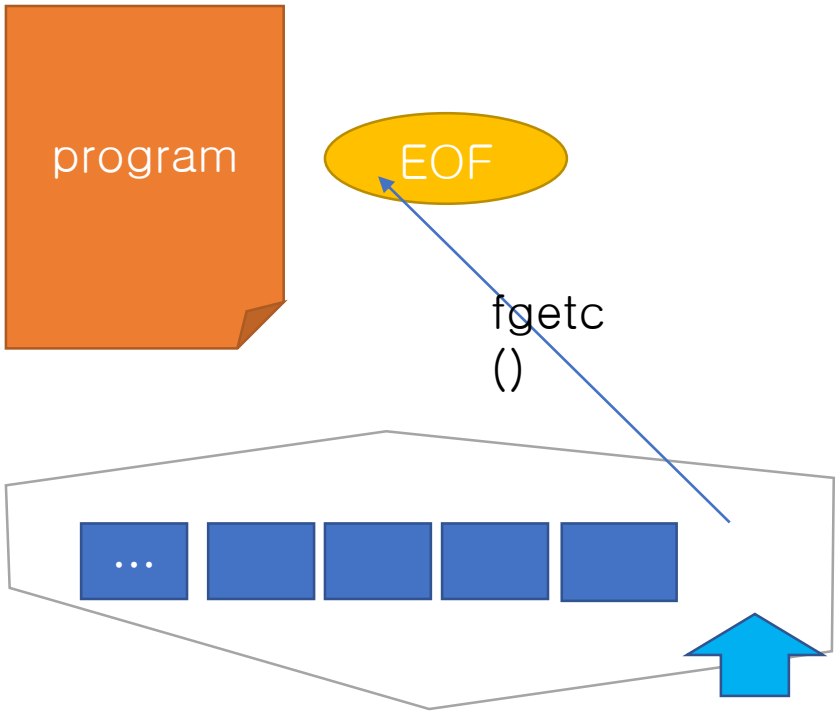




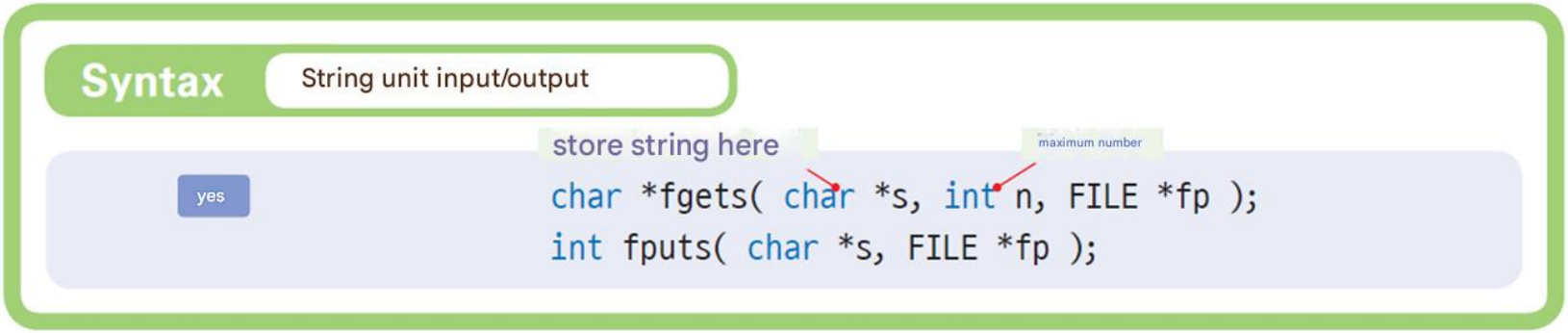
EOF

- EOF(End Of File): A special symbol indicating the end of a file.

#define EOF (-1)



String unit input/output



How does fgets() know it's reading one line?
fgets(buffer, size, fp) reads characters
from a file until one of the following happens:

1. A newline character (₩n) is read
2. size - 1 characters have been read to leave space for the null terminator
3. End of File (EOF) is reached

So, it stops at the end of a line, meaning it reads one line at a time, including the newline ₩n.

String unit input/output

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

int main( void )
{
      FILE *fp1, *fp2;
      char file1[100], file2[100];
      char buffer[100];

      printf ( " original file name : " );
      scanf ( "%s" , file1);

      printf ( " copy file name : " );
      scanf ( "%s" , file2);

      // first The file Read In mode Open .
      if ( (fp1 = fopen (file1, "r" )) == NULL )
      {
            fprintf ( stderr , " original Cannot open file % s .\n" , file1);
            exit(1);
      }
```

String unit input/output

```
        // second The file write In mode Open .
        if ( (fp2 = fopen (file2, "w" )) == NULL )
        {
            fprintf ( stderr , " copy Cannot open file % s .\n" , file2);
            exit(1);
        }

         // first The file second To file Copy .
         while ( fgets (buffer, 100, fp1) != NULL )
                fputs (buffer, fp2);

         fclose (fp1);
         fclose (fp2);

         return 0;
}
```

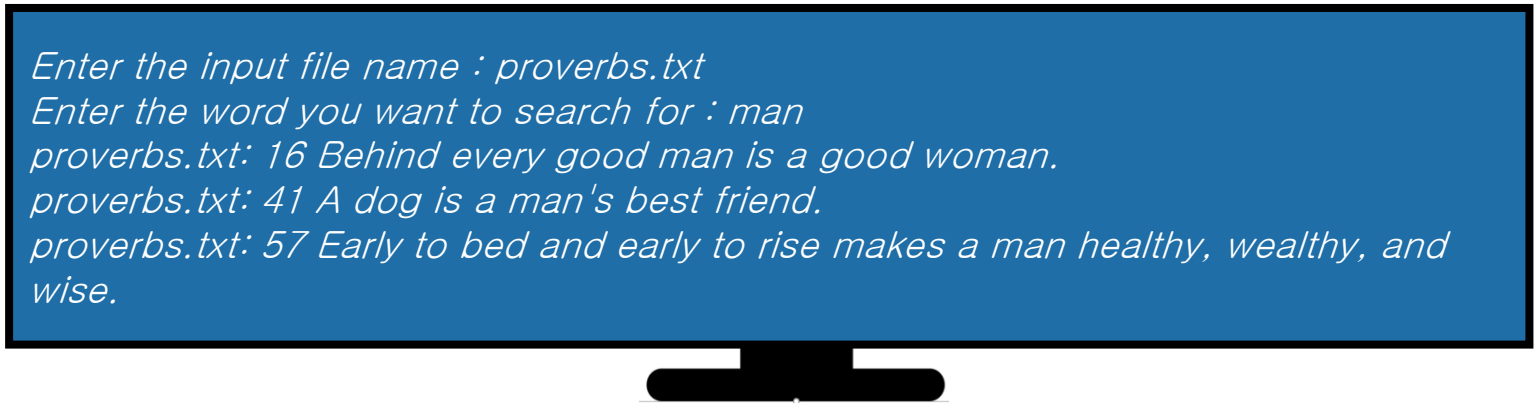
Original file name: a.txt
Copy file name: b.txt

Lab: Finding a specific string in a file

- Let's write a program that searches for a specific string in a text file. It takes the input text file name and the string to search for from the user.

입력 파일 proverbs.txt

```
Absence makes the heart grow fonder.
Actions speak louder than words.
All for one and one for all.
All's fair in love and war.
...
```



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

int main( void )
{
    FILE * fp ;
    char fname [128], buffer[256], word[256];
    int line_num = 0;

    printf ( " Enter the input file name : " );
    scanf ( "%s" , fname );

    printf ( " Enter the word to search : " );
    scanf ( "%s" , word);
```

```
// Open the file in read mode .
   if (( fp = fopen ( fname , "r" )) == NULL )
   {
      fprintf ( stderr , " Cannot open file %s .\n" , fname );
      exit(1);
   }

   while ( fgets (buffer, 256, fp )) {
      line_num ++;
      if ( strstr (buffer, word)) {
         printf ( "%s: %d word %s found .\n" , fname , line_num , word);
      }
   }
   fclose ( fp );

   return 0;
}
```

Formatted Input/Output

Syntax formatted input and output

yes

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

Example

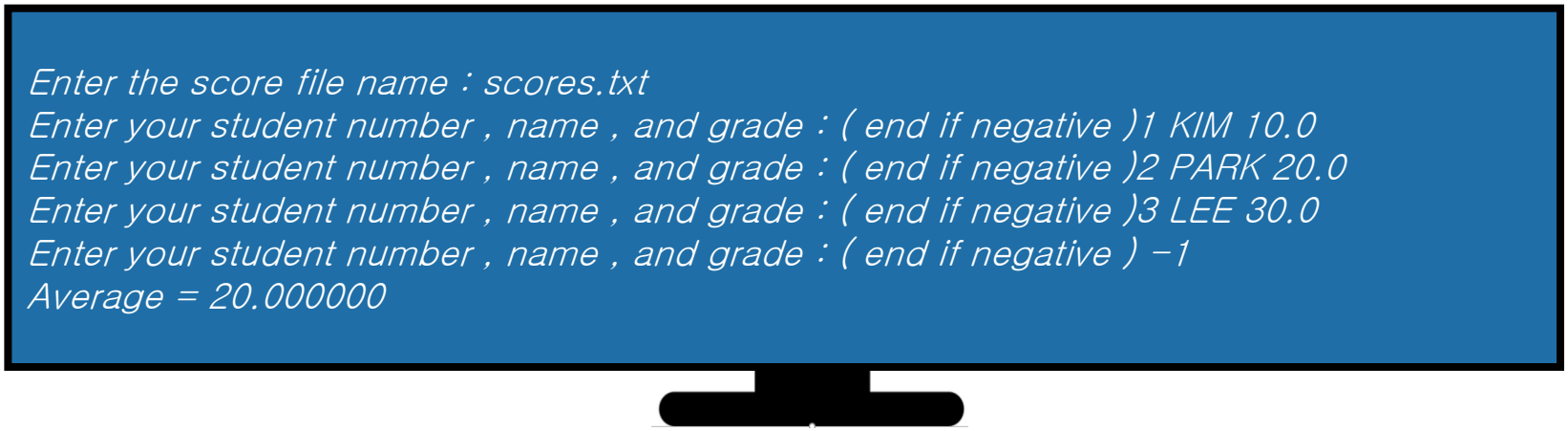
```
int main( void )
{
        FILE * fp ;
        char fname [100];
        int number, count = 0;
        char name[20];
        float score, total = 0.0;

        printf ( " Grade file Name Enter : " );
        scanf ( "%s" , fname );

        // Grades The file write In mode Open .
        if ( ( fp = fopen ( fname , "w" )) == NULL )
        {
             fprintf ( stderr , " grades Cannot open file % s .\n" , fname );
             exit(1);
        }
```

Example

```
// From the user Student number , name , and grades Input it In the file Save it.
while ( 1 )
{
    printf ( " Student number , name , grade Please enter : ( if negative end )" );
    scanf ( "%d" , &number);
    if ( number < 0 ) break
    scanf ( "%s %f" , name, &score);
    fprintf ( fp , " %d %s %f" , number, name, score);
}
fclose ( fp );
// Grades The file Read In mode Open .
if ( ( fp = fopen ( fname , "r" )) == NULL )
{
    fprintf ( stderr , " grades Cannot open file % s .\n" , fname );
    exit(1);
}
```

Example

```
        // from file Grades Read it The average Save .
        while ( ! feof ( fp ) )
        {
            fscanf ( fp , "%d %s %f" , &number, name, &score);
            total += score;
            count++;
        }
        printf ( " average = %f\n" , total/count);
        fclose ( fp );
        return 0;
}
```




```c
#include <stdio.h>
int writeFile( void )
{
    FILE * fp = NULL;

    fp = fopen ( "sample.txt" , "w" );
    if ( fp == NULL ) {
        printf ("file opening Failed \n" );
    }
    else {
        printf ("file opening Success \n" );
    }

    // 1. save character
    fputc ('a', fp );
    fputc ('b', fp );
    fputc ('c', fp );
    fputc ('\n', fp );

    // 2. save string
    fputs("hello", fp);
    fputs(" world\n", fp);

    // 3. save number
    fprintf(fp, "%d %d %d %.2f", 1, 2, 3, 3.14);

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

```c
int readFile() {
    FILE *fp = fopen("sample.txt", "r"); // read mode

    if (fp == NULL) {
        printf("Fail to open\n");
        return 1;
    }

    // 1. read 4 characters
    char ch1 = fgetc(fp);
    char ch2 = fgetc(fp);
    char ch3 = fgetc(fp);
    char ch4 = fgetc(fp);

    // 2. read string (12 character "hello world\n"
    // including white space)
    char str1[10];
    char str2[10];
    fscanf(fp, "%s %s", str1, str2);

    // 3. read 3 integers & 1 real number
    int n1, n2, n3;
    float f;
    fscanf(fp, "%d %d %d %f", &n1, &n2, &n3, &f);

    // print
    printf("\n== readFile ==\n");
    printf("Chars: %c %c %c %c", ch1, ch2, ch3, ch4);
    printf("String: %s %s\n", str1, str2);
    printf("Integers: %d %d %d\n", n1, n2, n3);
    printf("Float: %.2f\n", f);

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

file_text.c
continue...

```c
int readFileByOne( void )
{
    FILE * fp = NULL;
    int c;
    fp = fopen ( "sample.txt" , "r" );
    if ( fp == NULL )
        printf ( "file opening Failed \n");
    else
        printf ( "file opening Success \n");

    printf("\n== readFileByOne ==\n");
    while ((c = fgetc ( fp )) != EOF ) {
        putchar (c);
    }
    fclose ( fp );
    return 0;
}

int readFileByLine( void )
{
    FILE * fp = NULL;
    int SIZE = 100;
    char line[SIZE];

    fp = fopen ( "sample.txt" , "r" );
    if ( fp == NULL )
        printf ( "file opening Failed \n");
    else
        printf ( "file opening Success \n");

    printf("\n== readFileByLine ==\n");
    while (fgets (line, SIZE, fp )) {
        printf ("%s", line);
    }
    fclose ( fp );
    return 0;
}
```

```c
void main()
{
    writeFile();
    readFile();
    readFileByOne();
    readFileByLine();
}
```

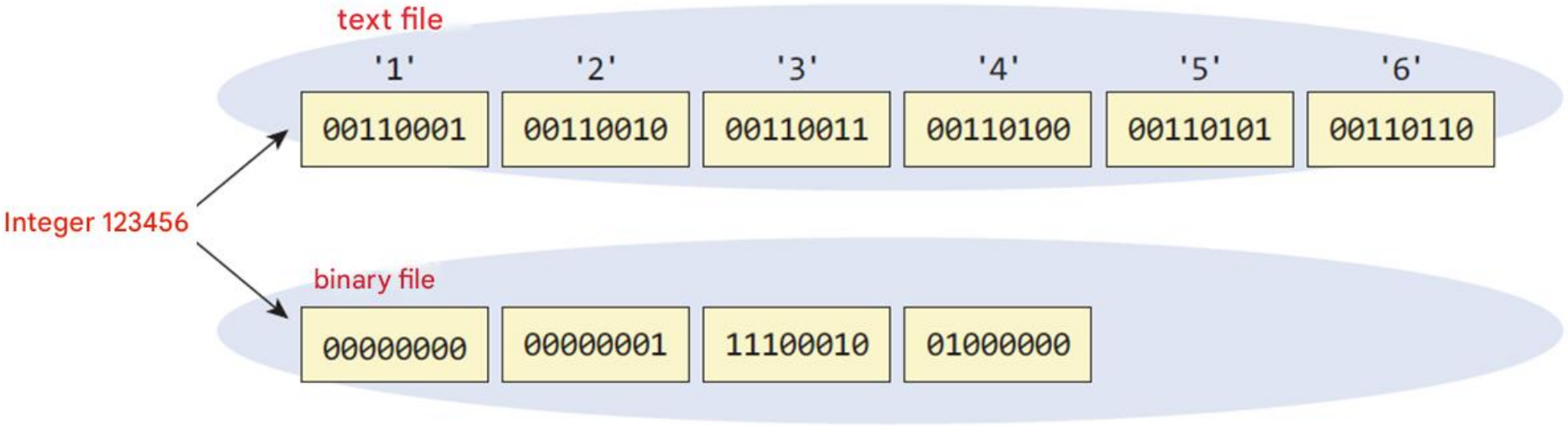
Return Value of fgets function

. Returns a pointer to a string if the read operation is successful.

. Returns NULL if it fails or reaches the end of the file.

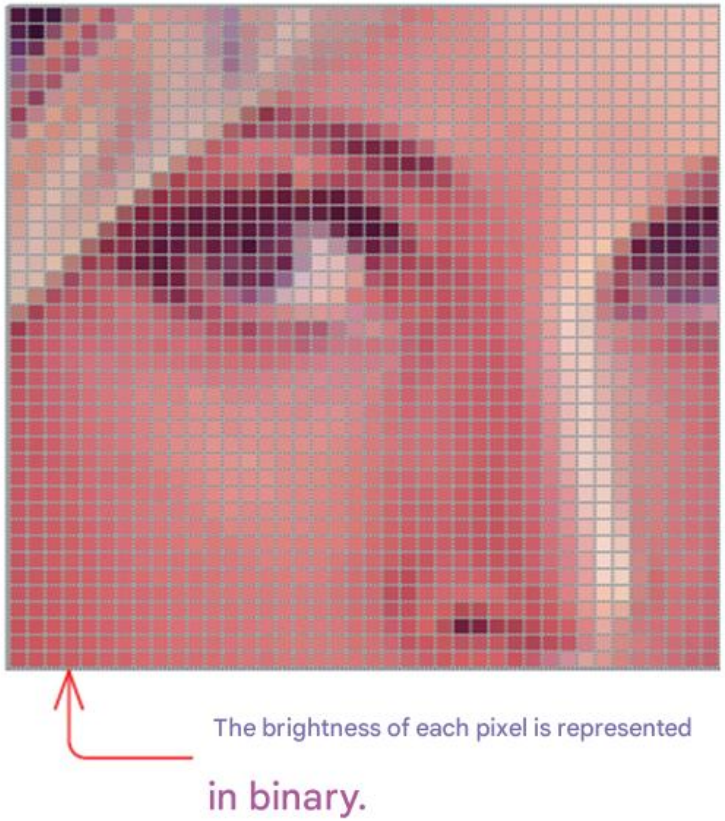
Writing and reading binary files

- Difference between text files and binary files
  - *Text file* : All data is converted to ASCII code and saved.
  - *Binary file* : Stores data exactly as it is represented on a computer.



Example of a binary file

- Image file or sound file



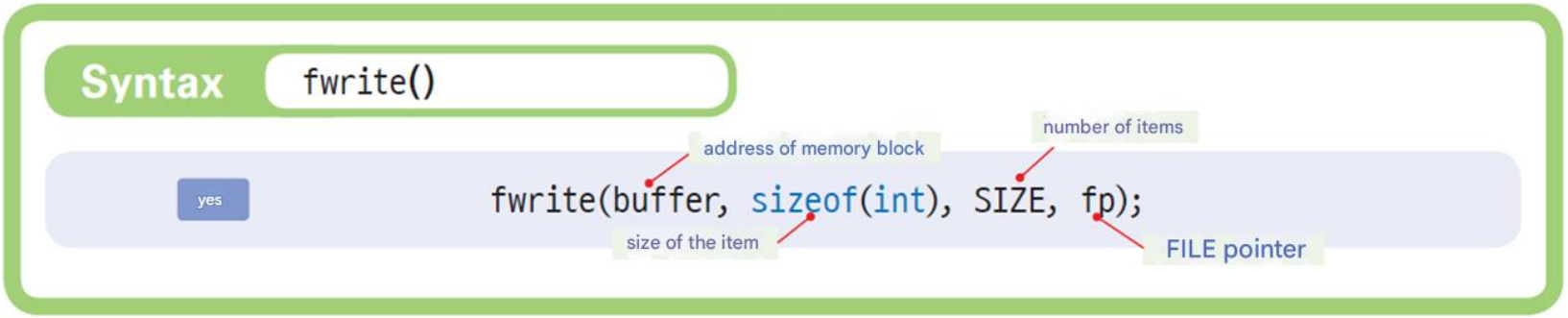
Write binary file

```
#include < stdio.h >

#define SIZE 5
int main( void )
{

    int buffer[ SIZE ] = { 10, 20, 30, 40, 50 };
    FILE * fp = NULL ;

    fp = fopen ( " binary.bin " , " wb " ); // ①
    if ( fp == NULL )
    {
        fprintf ( stderr , " binary.bin Cannot open file ." );
        return 1;
    }

    fwrite (buffer, sizeof ( int ), SIZE , fp ); // ②

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

Binary file mode

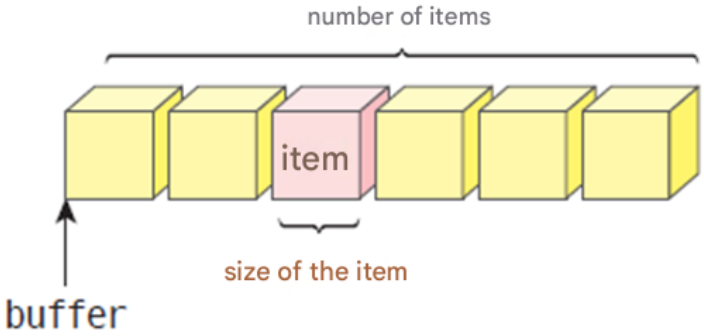
file mode	explanation
"rb"	Read mode + binary file mode
"wb"	Write mode + binary file mode
"ab"	Additional mode + binary file mode
"rb+"	Read and write mode + binary file mode
"wb+"	Write and read mode + binary file mode

Write binary file



fwrite (buffer, size, count, fp
)

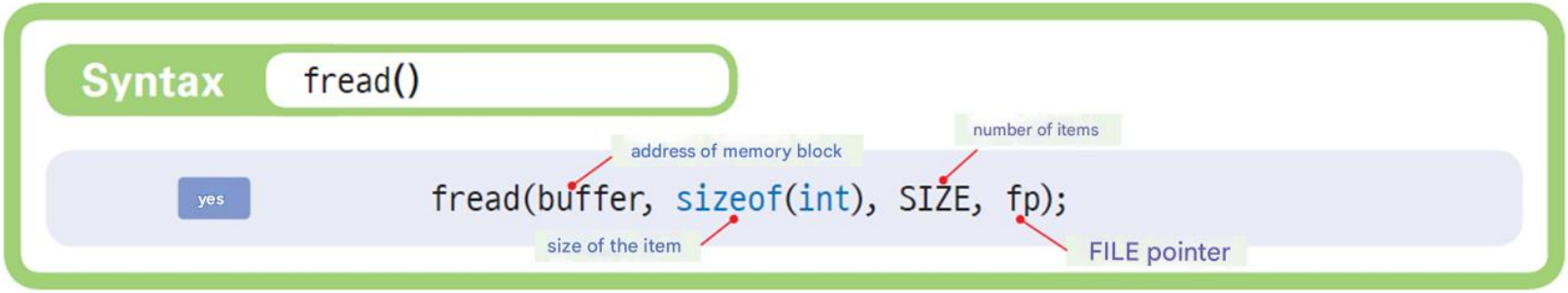
- buffer is the starting address of the memory block that contains the data to be written to the file .
- size is the size of the item being stored, in bytes .
- count is the number of items you want to store. If you want to write 10 int type data, the item size will be 4 and the number of items will be 10 .
- fp is a FILE pointer .



Reading binary files

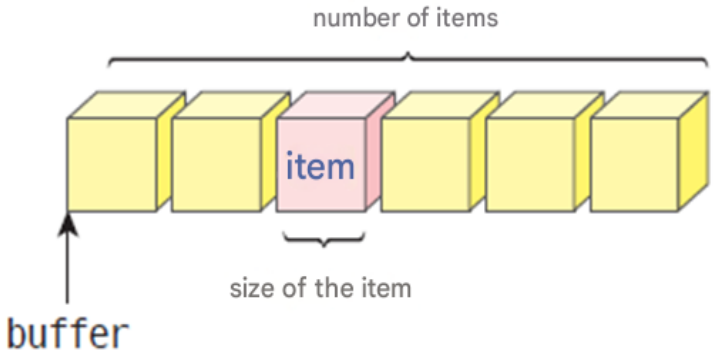
```
#include < stdio.h >
#define SIZE 5

int main( void )
{
    int i ;
    int buffer[ SIZE ];
    FILE * fp = NULL ;

    fp = fopen ( " binary.bin " , " rb " );
    if ( fp == NULL )
    {
        fprintf ( stderr , " binary.bin Cannot open file ." );
        return 1;
    }
    fread (buffer, sizeof ( int ), SIZE , fp );

    for (i = 0; i < SIZE ; i++)
        printf ( "%d " , buffer[ i ]);

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

Reading binary files



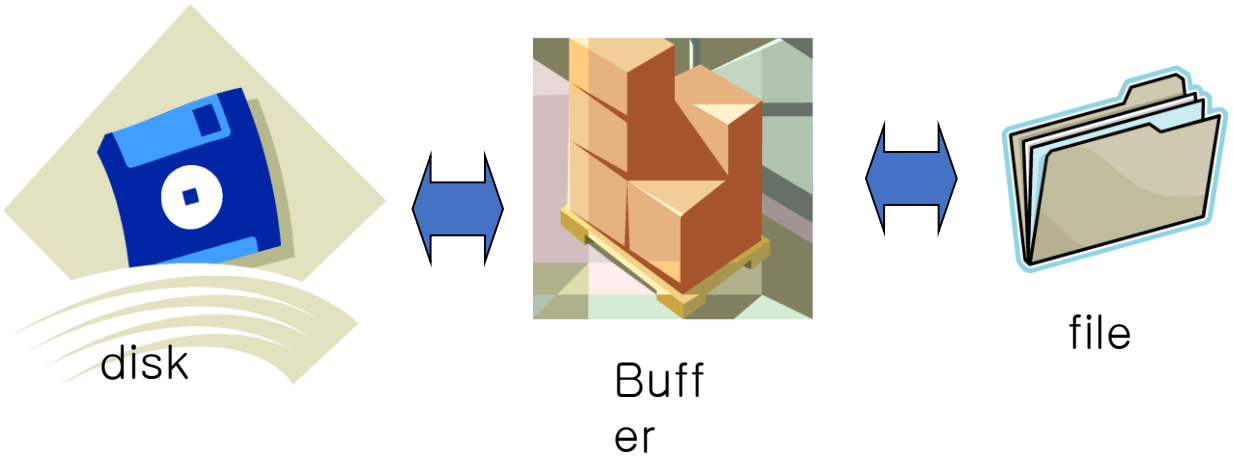
fread (buffer, size, count, fp)

- buffer is the starting address of the memory block that contains the data to be written to the file .
- size is the size of the item being stored, in bytes .
- count is the number of items you want to store . If you want to write 10 int type data, the item size will be 4 and the number of items will be 10 .
- fp is a FILE pointer .



Buffering

- A buffer is a block of memory used as a temporary storage location for data read from and written to a file.
- Since disk drives are block-unit devices, they operate most efficiently when input/output is performed in block units.
- Blocks of 1024 bytes are common.




```c
#include <stdio.h>

struct Data {
    char c;
    int i;
    float f;
};

int writeFile() {
    // "wb" = write binary
    FILE *fp = fopen("data.bin", "wb");
    if (fp == NULL) {
        printf("Fail to open\n");
        return 1;
    }

    struct Data d = { 'A', 100, 3.14f };

    // save binary of structure
    fwrite(&d, sizeof(struct Data), 1, fp);

    fclose(fp);
    printf("Complete writing.\n");
    return 0;
}
```

```c
int readFile() {
    // "rb" = read binary
    FILE *fp = fopen("data.bin", "rb");

    if (fp == NULL) {
        printf("Fail to open\n");
        return 1;
    }

    struct Data d;

    // read binary of structure
    fread(&d, sizeof(struct Data), 1, fp);

    printf("[read data]\n");
    printf("\tchar: %c\n", d.c);
    printf("\tint: %d\n", d.i);
    printf("\tfloat: %.2f\n", d.f);

    fclose(fp);
    return 0;
}

void main()
{
    int ret = writeFile();
    if(ret == 0){
        ret = readFile();

        if(ret == 0){
            printf("SUCCESS!!");
        } else {
            printf("FAIL!!");
        }
    }
}
```

Lab: Image Copy files

- Here, we will write a program that copies binary files .

```
Image file name : dog.jpg
Image file copied as copy.jpg
```

hint

- To read or write a binary file, append “ b” to the file mode when calling fopen () . To open a write-only file, use “ wb ” , and to open a read-only file, use “ rb ” .
  - src_file = fopen ("pome.jpg", " rb ");
  - dst_file = fopen ("copy.jpg", " wb ");

- To read data from a binary file, use fread () .
  - fread (buffer, 1, sizeof (buffer), src_file);

- To write data to a binary file, use fwrite () .
  - fwrite (buffer, 1, sizeof (buffer), dst_file);

- fread () returns the number of items successfully read , so if it returns 0 , it can be considered that the end of the file has been reached .

Example

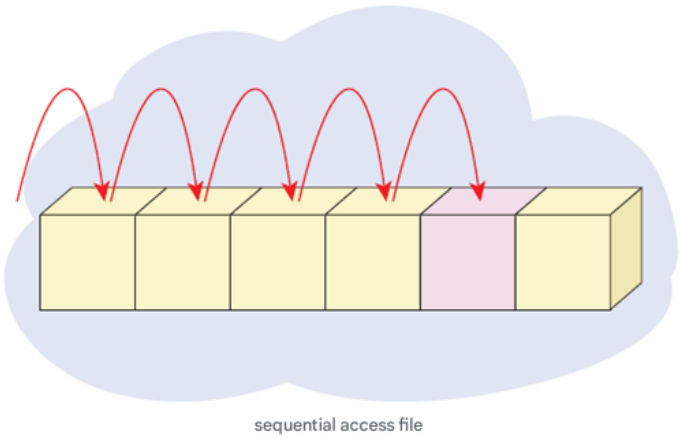
```
#include < stdio.h >

int main( void )
{
    FILE * src_file , * dst_file ;
    char filename[100];
    char buffer[1024];
    int r_count ;

    printf ( " Image file name : " );
    scanf ( "%s" , filename);

    src_file = fopen(filename, "rb" );
    dst_file = fopen ( "copy.jpg" , " wb " );
    if ( src_file == NULL || dst_file == NULL ) {
        fprintf ( stderr , " File open error \n" );
        return 1;
    }
```

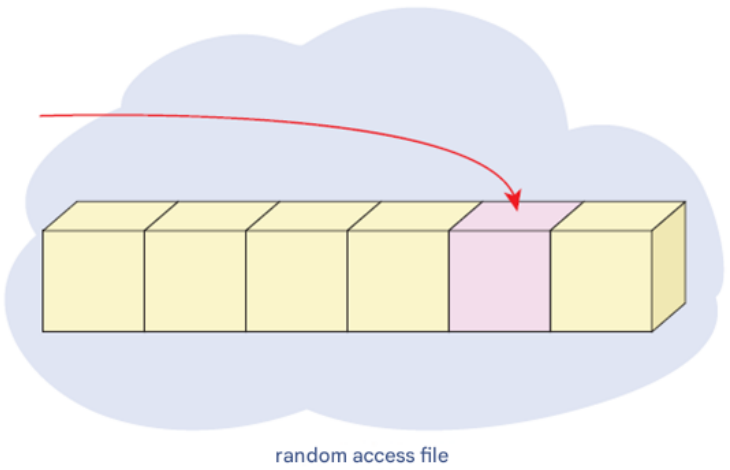
Example

```
   while (( r_count = fread (buffer, 1, sizeof (buffer), src_file )) > 0) {
       int w_count = fwrite (buffer, 1, r_count , dst_file );
       if ( w_count < 0) {
          fprintf ( stderr , " File writing error \n" );
          return 1;
       }
       if ( w_count < r_count ) {
          fprintf ( stderr , " Media write error \n" );
          return 1;
       }
    }
    printf ( " Image file copied as copy.jpg \n" );
    fclose ( src_file );
    fclose ( dst_file );
    return 0;
}
```

Random access files

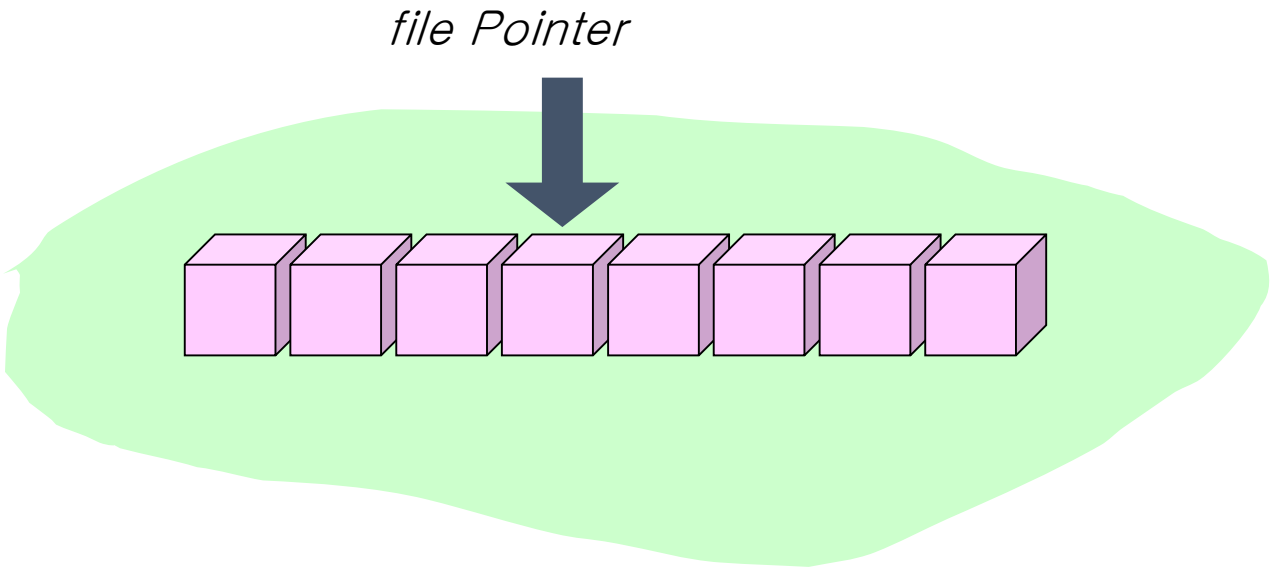
- **Sequential access** Method : How to read or write data sequentially from the beginning of a file
- **random access** Method : How to read and write from any location in the file





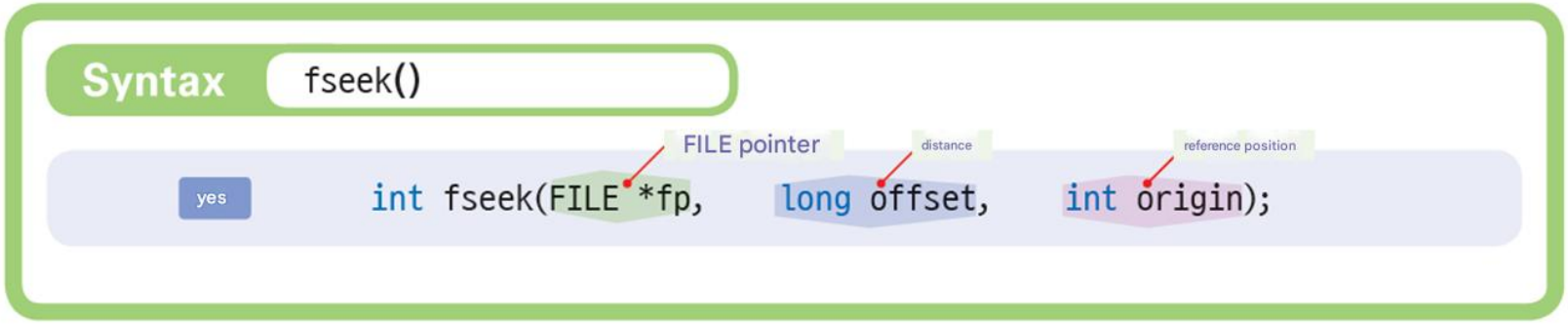
Principles of random access files

- File pointer : Indicates the current location of the file where read and write operations are taking place.



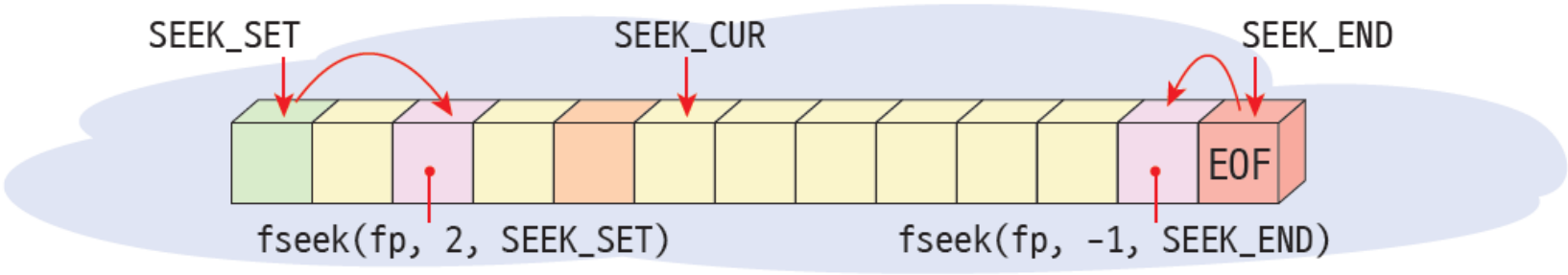
- Forcibly moving the file pointer allows random access

fseek ()



constant	value	explanation
SEEK_SET	0	start of file
SEEK_CUR	1	Current location
SEEK_END	2	end of file

fseek ()



- rewind (fp): Initializes the file pointer to the beginning .

ftell(), feof()

Syntax: ftell()

예

```
long ftell(FILE *fp);
```

Returns the current position of the file pointer

Syntax: foef()

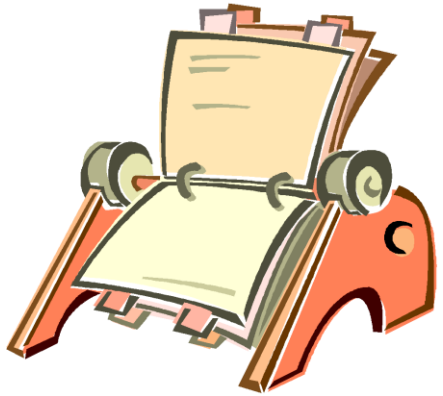
예

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

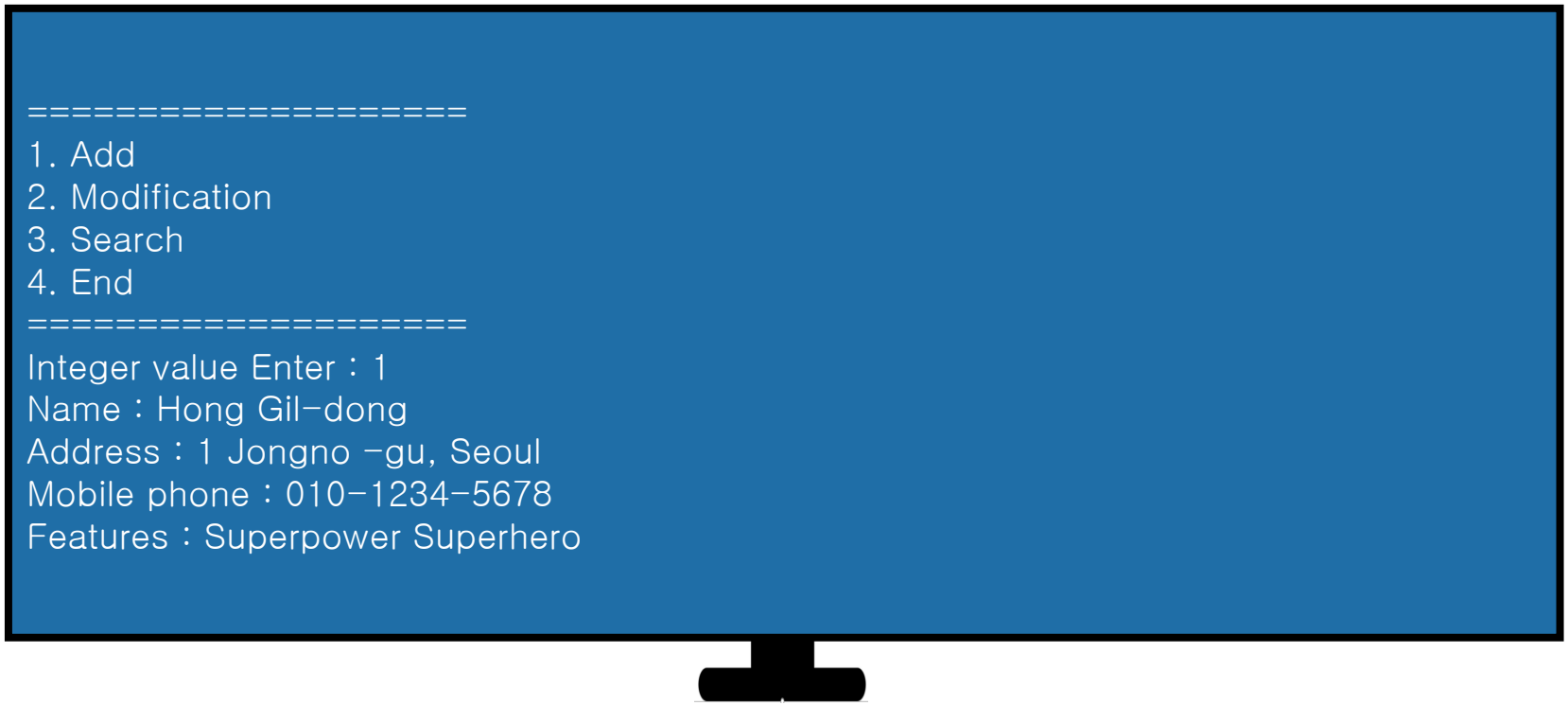
Returns whether the end of file has been reached

Mini Project: Creating an Address Book

- Let's write a simple program that can store and update information about yourself and your friends.
- Data entered or updated is saved as a file.
  Saved data can be searched .
- Let's create our own simple database system that will allow us to store various things we need.



Execution results



hint

1. The suitable one is “ a+” mode . It is mainly for adding and exploring.
2. You must always use fseek () before reading from a file .
3. When making modifications, it is better to create a new file and rewrite the entire thing there .

Example

```
#include < stdio.h >
#include < string.h >
#define SIZE 100
typedef struct person { // Represent contact information as a structure .
        char name[SIZE]; // name
        char address[SIZE]; // address
        char mobilephone [SIZE]; // mobile phone
        char desc [SIZE]; // Features
} PERSON;

void menu();
PERSON get_record ();
void print_record (PERSON data);
void add_record (FILE * fp );
void search_record (FILE * fp );
void update_record (FILE * fp );
```

```
int main( void )
{
	FILE * fp ;
	int select;
	// Open the binary file in append mode .
	if ( ( fp = fopen ( "address.dat" , "a+" )) == NULL ) {
		fprintf ( stderr , " Cannot open file for input );
		exit(1);
	}
	while (1) {
		menu(); // Display the menu
		printf ( " Enter an integer value : " ); // Get an integer from the user
		scanf ( "% d" ,&select );
		switch (select) {
			case 1: add_record ( fp ); break; // Add data
			case 2: update_record ( fp ); break; // Modify data
			case 3: search_record ( fp ); break; // Explore the data
			case 4: return 0;
		}
	}
	 fclose ( fp ); // Close the binary file
	 return 0;
}
```

```
// Receive data from the user and return it as a structure
PERSON get_record ()
{
	PERSON data;
	fflush ( stdin ); // Clear the standard input buffer
	printf ( " name ); gets(data.name); // Enter
	printf ( " address ); gets( data.address ); // Enter
	printf ( " cell phone ); gets( data.mobilephone ); // Enter
	printf ( " Features ); gets( data.desc ); // Receive features
	return data;
}
// Print the structure data to the screen .
void print_record (PERSON data)
{
	printf("Name\n", data.name);                    printf("Address\n" , data.address );
	printf("Mobile phone\n", data.mobilephone );   printf("Features\n" , data.desc );
}
```

```
// Function to display the menu on the screen
void menu()
{
	printf ( "====================\n" );
	printf ( " 1. Add \n 2. Modify \n 3. Search \n 4. End \n" );
	printf ( "====================\n" );
}
// Add data
void add_record (FILE * fp )
{
	PERSON data;
	data = get_record (); // Receive data from the user and store it in a structure
	fseek ( fp , 0, SEEK_END); // go to the end of the file
	fwrite (&data, sizeof (data), 1, fp ); // Write structure data to a file
}
```

```
// Explore the data
void search_record (FILE * fp )
{
	char name[SIZE];
	PERSON data;
	fseek ( fp , 0, SEEK_SET); // go to the beginning of the file
	fflush ( stdin );
	printf ( " The name of the person you want to search for " );
	gets(name); // Enter
	while (! feof ( fp )){ // repeat until the end of the file
		fread (&data, sizeof (data), 1, fp );
		if ( strcmp (data.name, name) == 0 ){ // Compare names
			print_record (data);
			break;
		}
	}
}
// Modify data
void update_record (FILE * fp )
{
 //...
}
```

Q & A