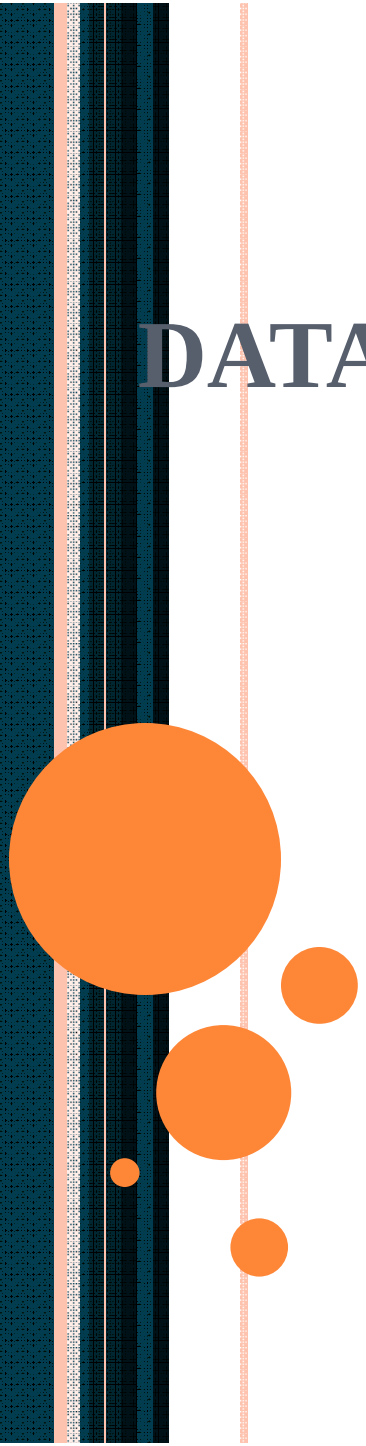
DATA STRUCTURES USING ‘C’



File Handling in C

Goals

By the end of this unit you should understand ...

- ... how to open a file to write to it.
- ... how to open a file to read from it.
- ... how to open a file to append data to it.
- ... how to read strings from a file.
- ... how to write strings to a file.

What is a File?

- A *file* is a collection of related data that a computers treats as a single unit.
- Computers store files to secondary storage so that the contents of files remain intact when a computer shuts down.
- When a computer reads a file, it copies the file from the storage device to memory; when it writes to a file, it transfers data from memory to the storage device.

Buffers

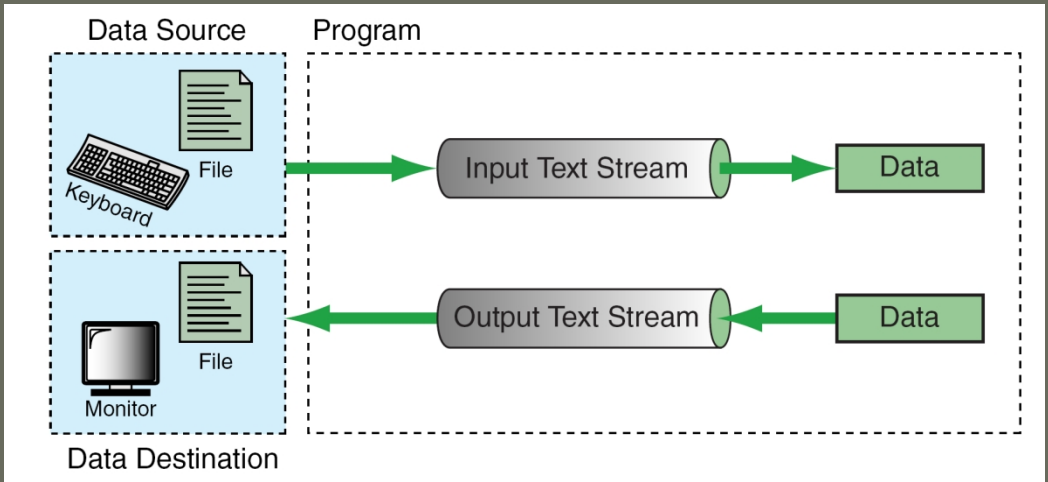
- A *buffer* is a “special work area” that holds data as the computer transfers them to/from memory.
- Buffers help to synchronize data the physical devices with the program.
- The physical requirements of the devices can deliver more data for input than a program can use at any one time. The buffer handles the overflow data until a program can use it.
- Moreover, the buffer also holds data until it is efficient to write that data to the storage device for output.

File Information Table

- A program requires several pieces of information about a file, including the name the OS uses for it, the position of the current character, etc.
- C uses a structure called **`FILE`** (defined in **`stdio.h`**) to store the attributes of a file.

Streams

- In C, we input/output data using streams. We can associate a stream with a device (i.e. the terminal) or with a file.
- C supports two types of files
  - Text Stream Files
  - Binary Stream Files



from Figure 7-1 in* Forouzan & Gilberg, *p. 395

Text Streams & Binary Streams

- *Text streams* consist of sequential characters divided into lines. Each line terminates with the newline character (`\n`).
- *Binary streams* consist of data values such as integers, floats or complex data types, “using their memory representation.”
- Today, we’ll concentrate solely on text streams …

Files & Streams

- A file is an “independent entity” with a name recorded by the operating system.
- A stream is created by a program.
- To work with a file, we must associate our stream name with the file name recorded by the OS.

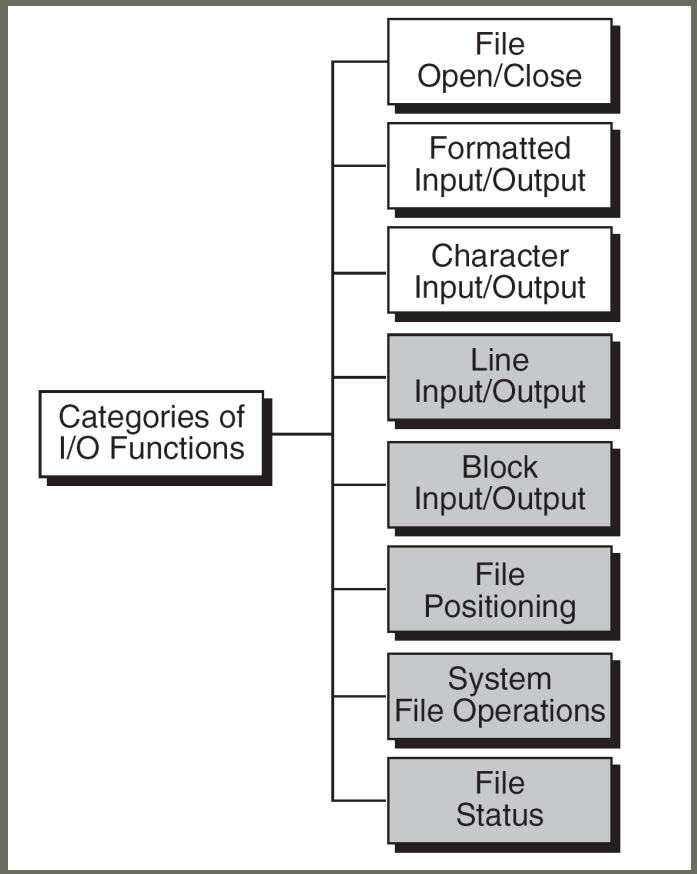
Steps in Processing a File

1. Create the stream via a pointer variable using the **`FILE`** structure:
   `FILE* spData;`
2. Open the file, associating the stream name with the file name.
3. Read or write the data.
4. Close the file.

System-Created Streams

- C automatically creates three streams that it opens and closes automatically for us in order to communicate with the terminal:
  - `stdin`
  - `stdout`
  - `stderr`
- We cannot re-declare these streams in our programs.

Standard I/O Functions in C



from Figure 7-2 in* Forouzan & Gilberg*, p. 398

File Open

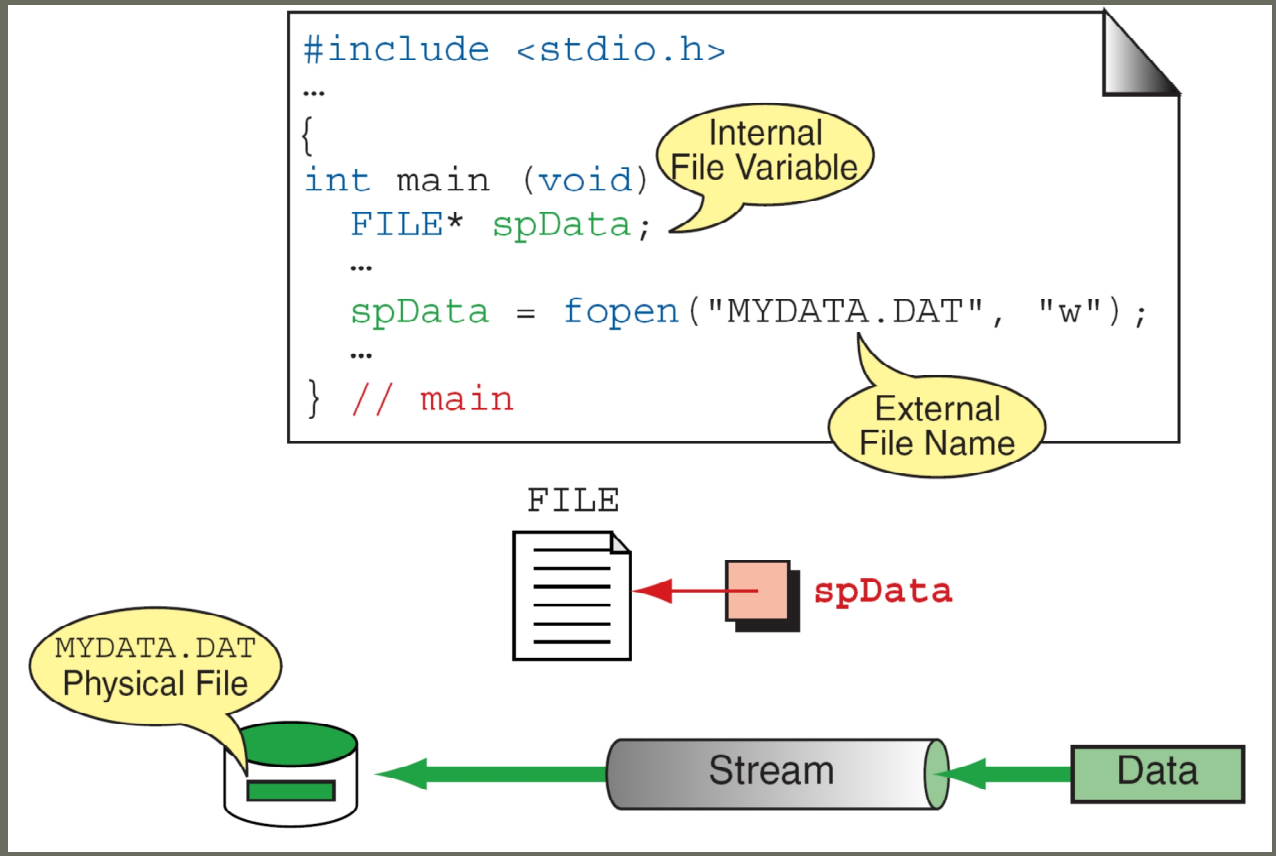
- The file open function (**`fopen`**) serves two purposes:
  - It makes the connection between the physical file and the stream.
  - It creates “a program file structure to store the information” C needs to process the file.
- Syntax:
  `fopen(“filename”, “mode”);`

More On `fopen`

- The *file mode* tells C how the program will use the file.
- The *filename* indicates the system name and location for the file.
- We assign the return value of `fopen` to our pointer variable:

```
spData = fopen(“MYFILE.DAT”, “w”);
spData = fopen(“A:\\MYFILE.DAT”, “w”);
```

More On `fopen`



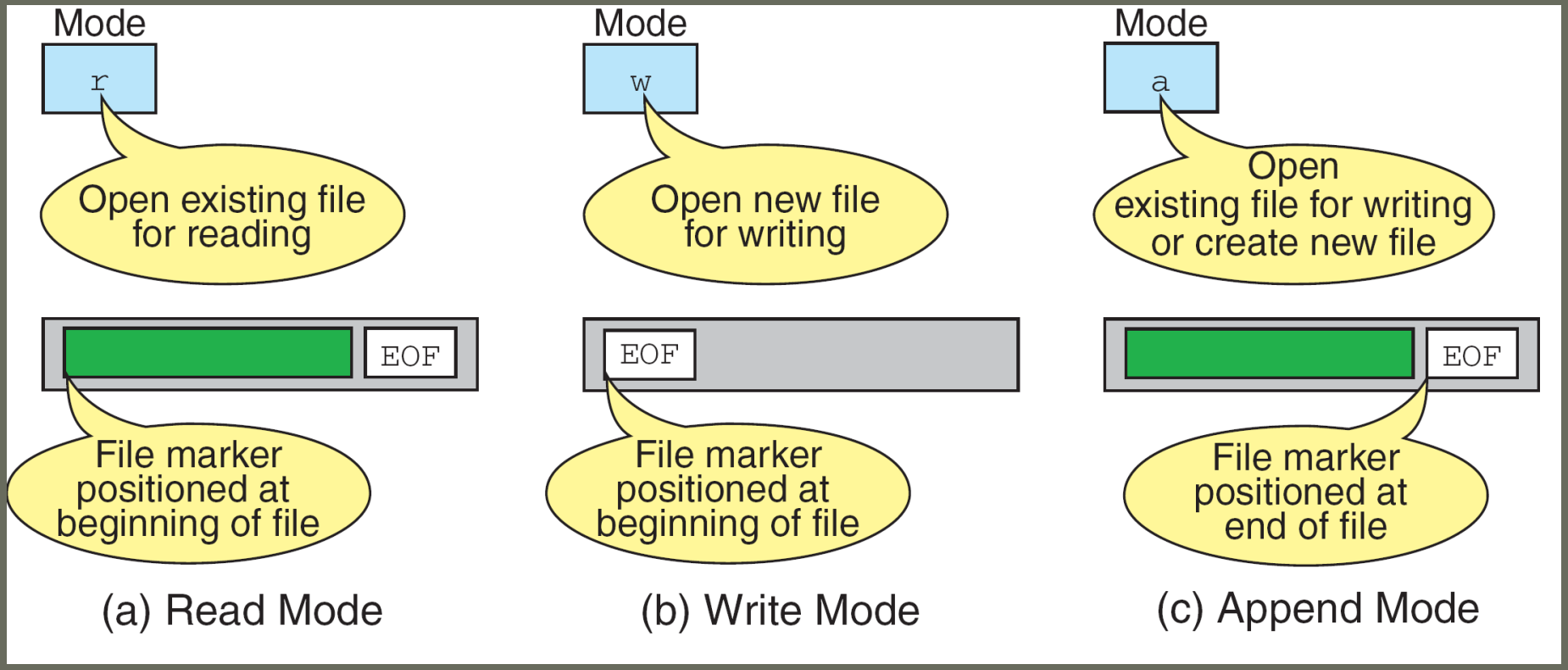
from Figure 7-3 in* Forouzan & Gilberg, *p. 399

File Open Modes

Mode	Meaning
r	Open text file in read mode • If file exists, the marker is positioned at beginning. • If file doesn't exist, error returned.
w	Open text file in write mode • If file exists, it is erased. • If file doesn't exist, it is created.
a	Open text file in append mode • If file exists, the marker is positioned at end. • If file doesn't exist, it is created.

from Table 7-1 in* Forouzan & Gilberg, *p. 400

More on File Open Modes



from Figure 7-4 in* Forouzan & Gilberg, *p. 401

Closing a File

- When we finish with a mode, we need to close the file before ending the program or beginning another mode with that same file.
- To close a file, we use **`fclose`** and the pointer variable:
  `fclose(spData);`

Code Example of `fopen`/`fclose`

- Example:

n305UsingFopenFclose.c

Additional I/O Functions

Terminal Input/Output

```
scanf ("control string", ...);
printf("control string", ...);
```

General Input/Output

```
fscanf (stream_pointer, "control string", ...);
fprintf(stream_pointer, "control string", ...);
```

from Table 7-2 in **Forouzan & Gilberg,** ***p. 403***

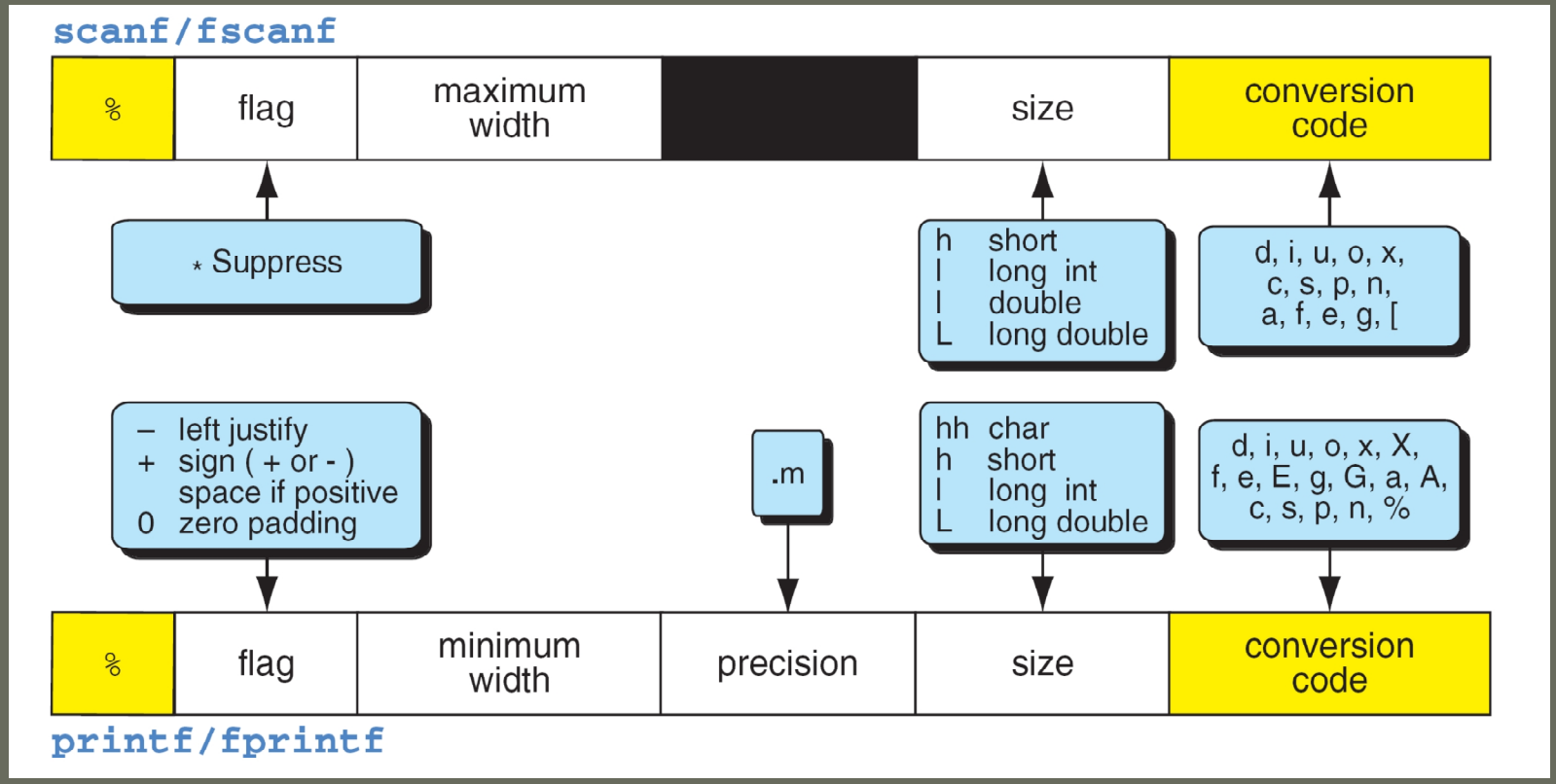
Whitespace in Format Control Strings

- For input, one or more whitespace characters in a format control string cause C to discard leading whitespace characters.
- For output, C copies whitespace characters in a format control string to the output stream.

Text in Format Control Strings

- For input, text must match exactly in the format control string to that of the input stream.
- For output, C copies text in the format control string to the output stream.

Conversion Specifications



from Figure 7-5 in* Forouzan & Gilberg, *p. 406

Conversion Specifications

- **“The number, order, and type of the conversion specifications must match the number, order, and type of the parameters in the list. Otherwise, the result will be unpredictable and may terminate the input/output function.”**

Input Data Formatting

- **`fscanf`**/**`scanf`** will process input characters until one of the following happens:
  - The function reaches the EOF indicator.
  - The function encounters an inappropriate character.
  - The function reads in a number of characters explicitly programmed as a maximum width field.

`fscanf`/`scanf` Flag & Width

- We use only one flag with fscanf/scanf – the supression flag (`*`), which tells the function to read input and then discard it:
```
scanf("%d %*c %f", &x, &y);
```
- The width is an optional modifier that with specify the maximum width for input (in characters):
```
scanf("%3d%2d%4d", &ssn1, &ssn2,
                   &ssn3);
```

Size & Conversion Codes

Argument Type	Size Specifier	Code
integral	hh (char), h (short), none (int), l (long), ll (long long)	i
integer	h (short), none (int), l (long). ll (long long)	d
unsigned int	hh (char), h (short), none (int), l (long), ll (long long)	u
character octal	hh (unsigned char)	o
integer hexadecimal	h (short), none (int), l (long), ll (long long)	x
real	none (double), l (double), L (double)	f
real (scientific)	none (double), l (double), L (double)	e
real (scientific)	none (double), l (double), L (double)	g
real (hexadecimal)	none (double), l (double), L (double)	a

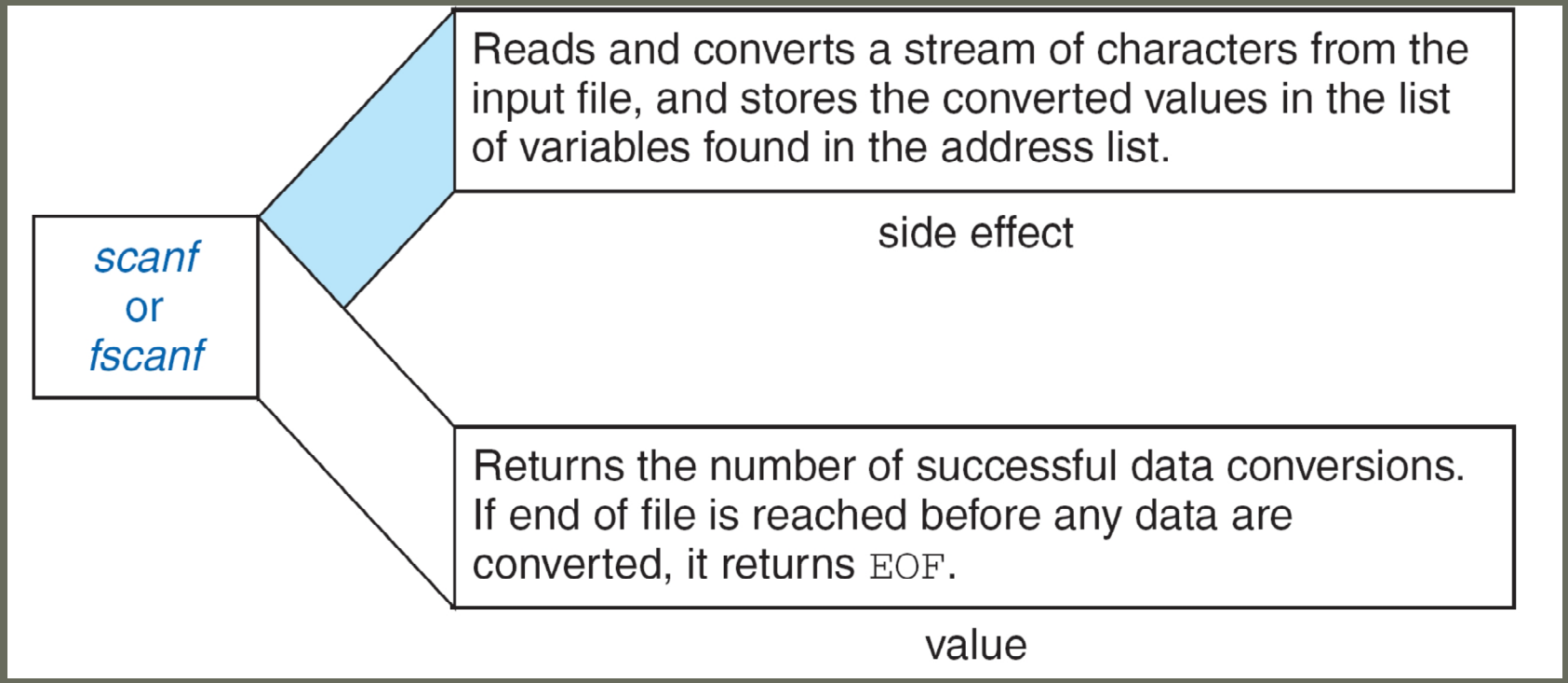
from Table 7-3 in* Forouzan & Gilberg, *p. 407

Size & Conversion Codes

Argument Type	Size Specifier	Code
character	none (char), l (wchar_t)	c
string	none (char string), l (wchar_t string)	s
pointer		p
integer (for count)	none (int), hh (char), h (short), l (long), ll (long long)	n
set	none (char), l (wchar_t)	[

from Table 7-3 in* Forouzan & Gilberg, *p. 408

Side Effect & Value of `fscanf`/`scanf`



from Figure 7-6 in* Forouzan & Gilberg, *p. 410

Input Stream Issues

1. There is always a return character at the end of an input stream due to the fact that C buffers the stream.
2. `fscanf`/`scanf` functions leave the return character in the buffer. To force a discard of the character, begin your format control string with a space character.
3. `fscanf`/`scanf` terminate when all specified operations in the control string complete; if the control string ends with a whitespace character, `fscanf`/`scanf` continue (they terminate only with a non-whitespace control string).

`fprintf`/`printf` Flags, Sizes & Conversion Codes

Argument Type	Flag	Size Specifier	code
integer	-, +, 0, space	hh (char), h (short), none (int), l (long), ll (long long)	d, i
unsigned integer	-, +, 0, space	hh (char), h (short), none (int), l (long), ll (long long)	u
integer (octal)	-, +, 0, #, space	hh (char), h (short), none (int), l (long), ll (long long)	o
integer (hex)	-, +, 0, #, space	hh (char), h (short), none (int), l (long), ll (long long)	x, X
real	-, +, 0, #, space	none (double), l (double), L (double)	f
real (scientific)	-, +, 0, #, space	none (double), l (double), L (double)	e, E
real (scientific)	-, +, 0, #, space	none (double), l (double), L (double)	g, G

from Table 7-4 in* Forouzan & Gilberg, *p. 419

`fprintf`/`printf` Flags, Sizes & Conversion Codes

real (hexadecimal)	-, +, 0, #, space	none (double), l (double), L (double)	a, A
character	-	none (char), l (w-char)	c
string	-	none (char string), l (w-char string)	s
pointer			p
integer (for count)		none (int), h (short), l (long)	n
to print %			%

from Table 7-4 in* Forouzan & Gilberg, *p. 419

`fprintf`/`printf` Output Flags

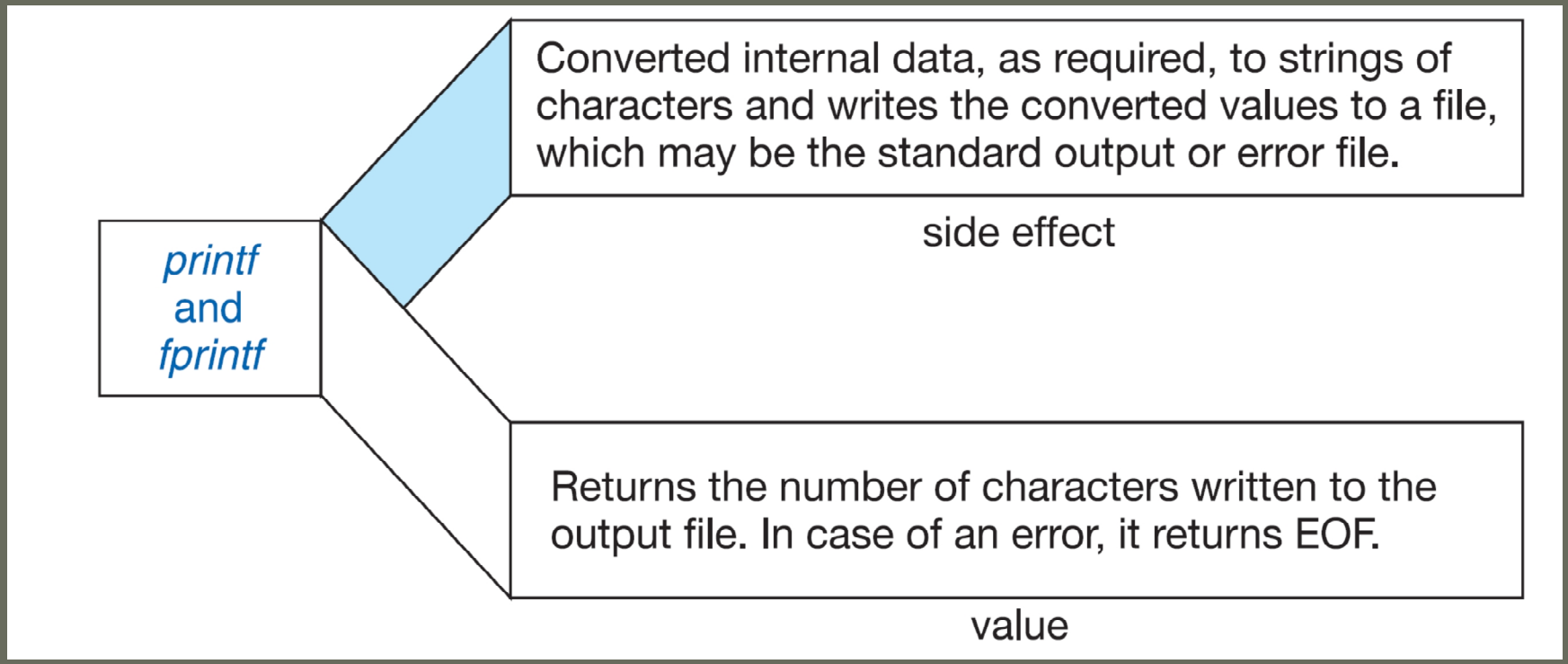
Flag Type	Flag Code	Formatting
Justification	none	right justified
	–	left justified
Padding	none	space padding
	0	zero padding
Sign	none	positive value: no sign negative value: –
	+	positive value: + negative value: –
	space	positive value: space negative value: –
Alternate	#	print alternative format for scientific, hexadecimal, and octal.

from Table 7-5 in* Forouzan & Gilberg, *p. 419-420

Width & Precision in Output

- Width for output specifies a *minimum width*. If data are wider, C will print all the data.
- We specify precision with a period followed by an integer:
  - For integers, precision specifies the minimum number of digits to print (incl. leading zeroes).
  - For floating-point numbers, precision specifies the number of digits to print to the right of the floating point.
  - For scientific numbers (g and G), precision specifies how many significant digits to print.

Output Side Effect & Value



from Figure 7-11 in* Forouzan & Gilberg, *p. 423

Code Example of `fscanf`

- Example:

n305UsingFscanf.c

Code Example of `fprintf`

- Example:

n305UsingFprintf.c

Code Example of Append Mode

- Example:

n305AppendMode.c

Code Example of File Handling

- Example:

n305StudentGrades.c

Code Example of String Input

- Example:

n305StringInput.c

Code Example of String Output

- Example:

n305StringOutput.c

Resources

- Forouzan, Behrouz & Richard Gilberg, *Computer Science: A Structured Programming Approach Using C*. Thomson Course Technology: 2007.