7. Text Input/Output

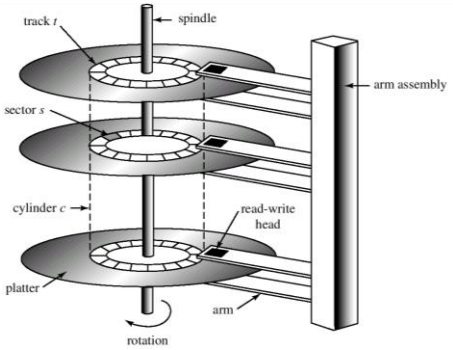
C Programming

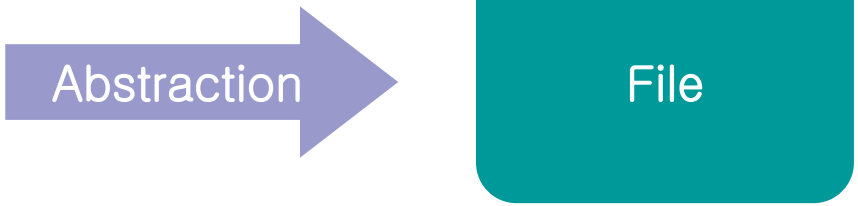
Agenda

- File and Stream
- File Open/Close
- Formatting Input/Output Functions
- Character Input/Output Functions

Files

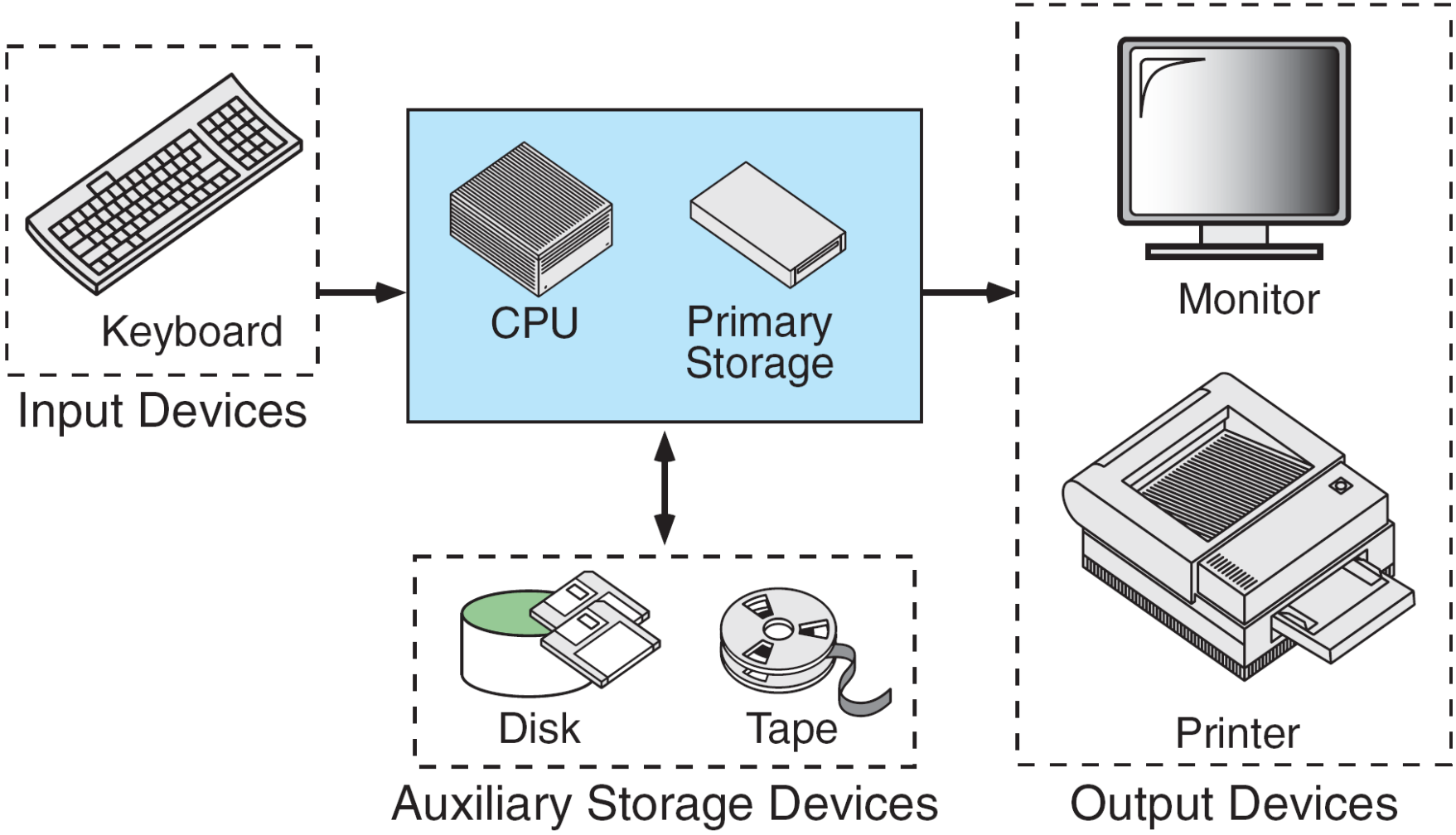
- File: external collection of related data treated as a unit
  - Memory space is abstracted through variable
  - Secondary storage space is abstracted through file
    - Magnetic disk, optical disk, magnetic tape, …





- Why file?
  - Cheap, large space, non-volatile

Computer Hardware



Files

- Each file is identified by filename or pathname
  - Filename

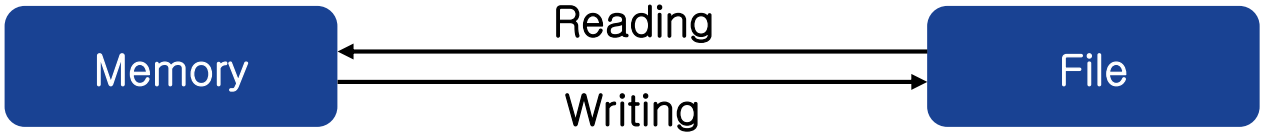
    Ex) hello.c
  - Pathname = directory name + filename

    Ex) /home/user/hello.c (UNIX)

    C:₩Source₩hello.c (Windows)

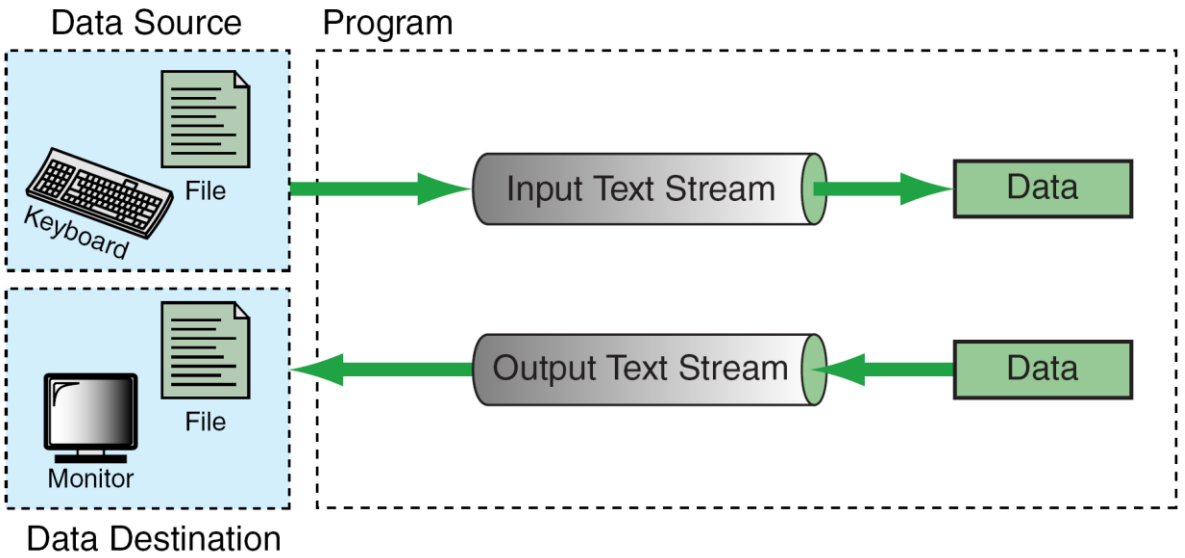
    → In C language "C:₩₩Source₩₩hello.c"

- File access
  - Reading: data moves from file to memory
  - Writing: data moves from memory to file



Streams

- In C, data is read and written through stream.
  - Stream can be associated with terminal, file, and other data sources or destinations



Streams

- C uses two types of streams
  - Text stream: consist of sequence of characters
    Ex) source file
  - Binary stream: consists a sequence of data values
    Ex) execution file, database file, …
    - Explained in chapter 13
- Stream-file processing
  - File: entity managed by OS
  - Stream: entity created by program

→ To access a file, we must associate a stream with a file

File Access

- File operations
  1. Declare pointer of a stream;
     Ex) FILE *spData = NULL;

  2. Open a file
     - Associate stream to a file

     Ex) spData = fopen("myfile.txt", "r");

  3. Access file
     - Read or write file through stream

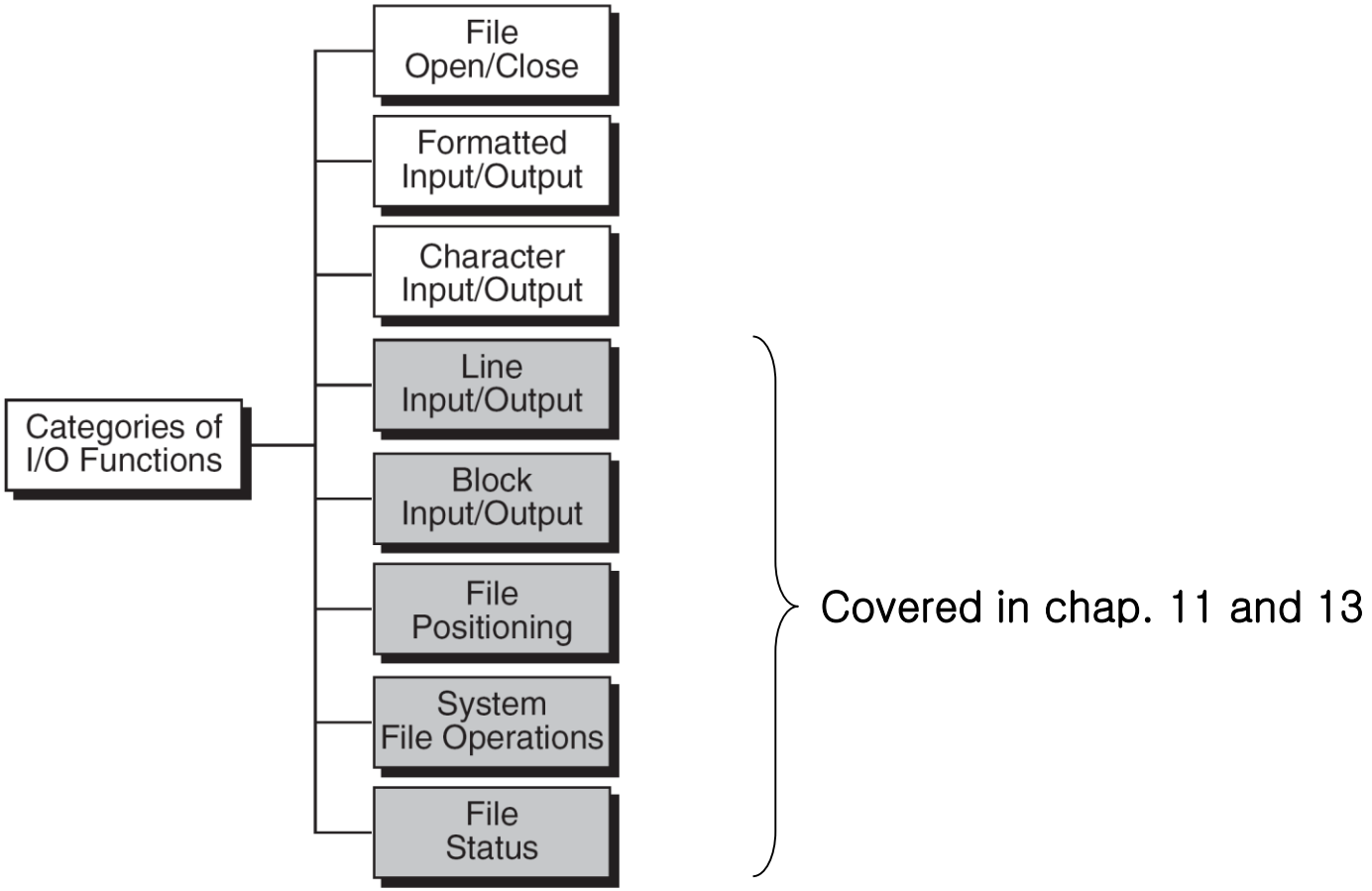
     Ex) fscanf(spData, "%d", &value);

  4. Close a file
     - Breaks association between stream and file

     Ex) fclose(spData);

Standard Library Input/Output Functions

- Declared in stdio.h



Agenda

- File and Stream
- File Open/Close
- Formatting Input/Output Functions
- Character Input/Output Functions

File Open

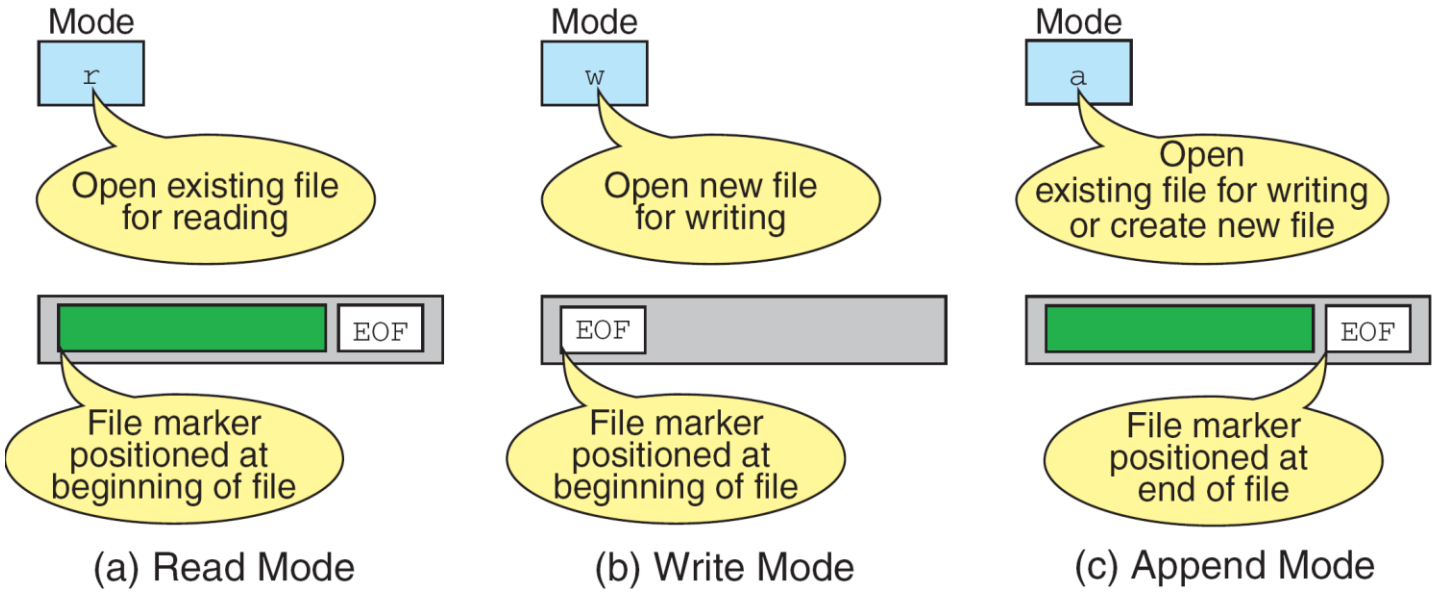
- File open: prepares a file for processing
  - Syntax: FILE* fopen("filename", "mode");
    - filename: name of physical file
    - Mode: string to indicate how the file will be used
    - Return value: pointer to a stream (FILE*)
      - If it fails to open a file, return NULL.

Ex) FILE* spData = fopen("MYFILE.DAT", "w");

FILE* spData = fopen("A:₩₩MYFILE.DAT", "w");

File Open

- File open modes
  - r: read mode, w: write mode, a: append mode



File Open

- 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.

- Combinations
  - "rw", "ra", "rb", "rab", …
    - b: binary mode

File Close

- File close: break the association with file and free system resources such as buffer space
  - Syntax: fclose(stream_pointer);

File Open/Close Errors

- What happens if fopen() or fclose() fails?

  Ex) specified file does not exists

  - fopen() returns NULL
  - fclose() returns EOF (-1)

- Checking errors

  Ex)

```
if((spTemps = fopen(filename, “r”)) == NULL){
    printf(“Error opening %s₩n”, filename);
    exit(100);
}
…
…
if(fclose(spTemps) == EOF){
    printf(“Error closing %s₩n”, filename);
    exit(102);
}
```

Agenda

- File and Stream
- File Open/Close
- Formatting Input/Output Functions
- Character Input/Output Functions

Formatting Input/Output

- Formatting input/output functions
  - Keyboard and monitor: *printf*, *scanf*
  - File: *fprintf*, *fscanf*

Terminal Input/Output

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

General Input/Output

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

Usage of *fprintf* and *fscanf* is very similar to that of *printf* and *scanf*

Formatting Input/Output Functions

- Formatting input

  FILE *spIn = fopen("file name", "r");

  fscanf(spIn, "format string", address list);

- Formatting output

  FILE *spOut = fopen("file name", "w");

  fprintf(spOut, "format string", value list);

- System created stream (declared in stdio.h)
  - fscanf(stdin, "format string", address list);
  - fprintf(stdout, "format string", value list);
  - fprintf(stderr, "format string", value list);

Additional Conversion Specifiers

- Conversion code
  - Decimal: %d
  - Octal: %o
  - Hexadecimal: %x
  - Scientific notation: %e, %g, %a
    - Significant + exponent

    Ex) float x =1234567; // $1.234567*10^6$

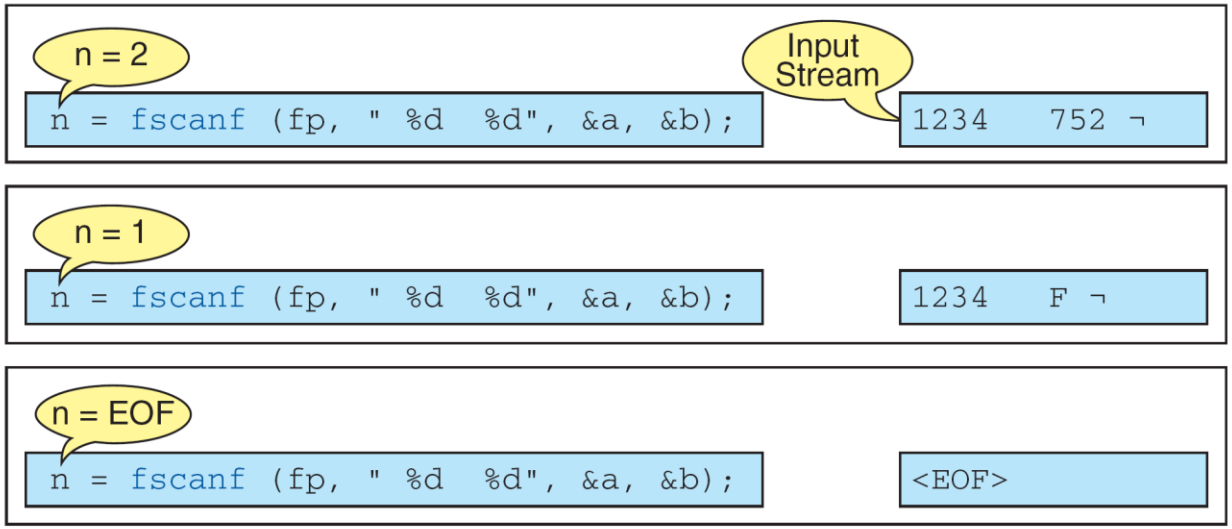
    printf("%e", x); // result: 1.234567e+06
  - Pointer (address): %p

    Ex) int i = 100;

    printf("%d, %p₩n", i, &i); // 100, 0022ff6c

Return Values of Formatting Input Functions

- Return value of input functions (*scanf*, *fscanf*)
  - # of successful data conversions



  - If end of file is reached before any data are converted, return value is EOF

Checking Formatting Input Errors

- Result of formatting input function should be checked

```
ioResult = scanf("%d%f", &amount, &price);
if(ioResult != 2) {
    printf("error in scanf, ioResult = %d₩n", ioResult);
    ...
}
```

Example: Copy Text File of Integer

- Copy a text file containing integers

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

#define InputFile "P07-03.DAT"
#define OutputFile "P07-04.DAT"

int main()
{
   FILE *in = NULL;
   FILE *out = NULL;

   in = fopen(InputFile, "r");
   out = fopen(OutputFile, "w");
   if(in == NULL || out == NULL){
      // display error message and terminates
      exit(-1);
   }
   while(fscanf(in, "%d", &num) == 1)
      fprintf(out, "%d ", num);

   fclose(in);
   fclose(out);

   return 0;
}
```

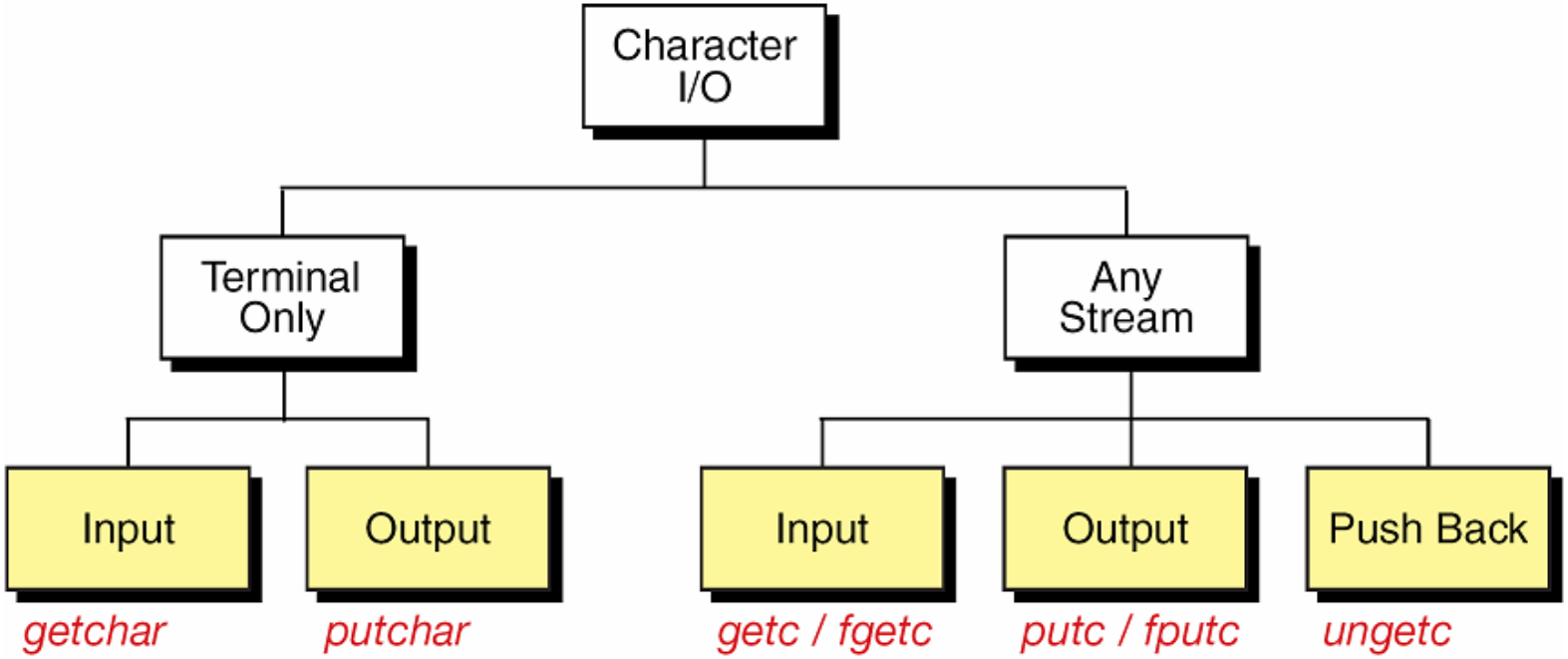
Content of P07-03.DAT:
1 2 3 4 5 6 7 8 9 10

Agenda

- File and Stream
- File Open/Close
- Formatting Input/Output Functions
- Character Input/Output Functions

Character Input/Output Functions

- Character I/O function: read or write single character at a time



Terminal Character I/O

- Read a character: getchar
  - Syntax: int getchar(void);

    Ex) char c = 0;

    c = getchar(); // ≈ scanf("%c", &c)
    - Why return type is integer? Because EOF is (int)-1. (stddef.h)

- Write a character: putchar
  - Syntax: int putchar(int ch);

    Ex) char c = 'a';

    putchar(c);
    - Return value
      - Success: the character written
      - Error: EOF

Terminal and File Character I/O

- Reading a character: getc and fgetc
  - int getc(FILE *spIn); // same as fgetc
  - int fgetc(FILE *spIn);

Ex)

```
char c = 0;
FILE *fp = fopen("myfile.txt", "r");
...
c = fgetc(fp);
...
fclose(fp);
```

Character Input Functions

- Character input functions
  - int getchar(void);
  - int getc(FILE *spIn);
  - int fgetc(FILE *spIn);
  - int getch(void);
    - Enter is not required to read a character
    - Not a standard function (declared in conio.h)

Terminal and File Character I/O

- Write a character: putc and fputc
  - int putc(int ch, FILE* spOut); // same as fputc
  - int fputc(int ch, FILE* spOut);

Ex)

```
char c = 'a';
FILE *fp = fopen("myfile.txt", "w");
...
fputc(c, fp);
...
fclose(fp);
```

Example: Creating Text File

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

#define FileName "text.txt"

int main()
{
	char c = 0;
	FILE *fp = fopen(FileName, "w");

	if(fp == NULL){
		printf("Failed to open %s₩n", FileName);
		exit(-1);
	}

	printf("Write a text to store in %s₩n", FileName);
	printf("Press CTRL-Z to terminate.₩n");

	while((c = getchar()) != EOF)
		fputc(c, fp);

	fclose(fp);

	return 0;
}
```

Example: Copying Text File

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

#define SrcFile "text.txt"
#define DestFile "text_copy.txt"

int main()
{
	char c = 0;
	FILE *inFP = fopen(SrcFile, "r");
	FILE *outFP = fopen(DestFile, "w");

	if(inFP == NULL || outFP == NULL){
		printf("Failed to open files₩n");
		exit(-1);
	}

	printf("This program copies %s to %s₩n", SrcFile, DestFile);
	while((c = fgetc(inFP)) != EOF)
		fputc(c, outFP);

	fclose(inFP);
	fclose(outFP);

	return 0;
}
```

Other Examples

- Counting characters and lines in a text file
- Counting words in a text file