Chapter 9. IO In C

Byoung-Tak Zhang
TA: Hanock Kwak

Biointelligence Laboratory
School of Computer Science and Engineering
Seoul National Univertisy

http://bi.snu.ac.kr

The Output Function printf()

- `int printf ( const char * format, ... );`
- Writes the C string pointed by *format* to the standard output (**stdout**).
- Format specifier
  - It can optionally contain embedded *format specifiers* that are replaced by the values specified in subsequent additional arguments and formatted as requested.
    - e.g. **%d %f %c %s**

```
int a = 100;
printf("Decimal: %d Octa: %o Hexa: %x\n", a, a, a);
```

Format Specifier in printf()

- A *format specifier* follows this prototype
  - %[flags][width][.precision][length]specifier

specifier	Output	Example
a	Hexadecimal floating point, lowercase	-0xc.90fep-2
A	Hexadecimal floating point, uppercase	-0XC.90FEP-2
c	Character	a
s	String of characters	sample
p	Pointer address	b8000000
n	Nothing printed. The corresponding argument must be a poi nter to a signed int. The number of characters written so far is st ored in the pointed location.	
%	A % followed by another % character will wri te a single % to the stream.	%

specifier	Output	Example
d or i	Signed decimal integer	392
u	Unsigned decimal integer	7235
o	Unsigned octal	610
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (up percase)	7FA
f	Decimal floating point, lowercase	392.65
F	Decimal floating point, uppercase	392.65
e	Scientific notation (mantissa/expon ent), lowercase	3.9265e+2
E	Scientific notation (mantissa/expon ent), uppercase	3.9265E+2
g	Use the shortest representation: % e or %f	392.65
G	Use the shortest representation: % E or %F	392.65

Format Specifier in printf()

- Flags

flags	description
-	Left-justify within the given field width; Right justification is the default (see wid th sub-specifier).
+	Forces to preceed the result with a plus or minus sign (+ or -) even for positive numbers. By default, only negative numbers are preceded with a - sign.
(space)	If no sign is going to be written, a blank space is inserted before the value.
#	Used with o, x or X specifiers the value is preceeded with 0, 0x or 0X respecti vely for values different than zero. Used with a, A, e, E, f, F, g or G it forces the written output to contain a decim al point even if no more digits follow. By default, if no digits follow, no decimal point is written.
0	Left-pads the number with zeroes (0) instead of spaces when padding is speci fied (see width sub-specifier).

Format Specifier in printf()

- width

width	**description**
(number)	Minimum number of characters to be printed. If the value to be printed is short er than this number, the result is padded with blank spaces. The value is not tr uncated even if the result is larger.
*	The *width* is not specified in the *format* string, but as an additional integer valu e argument preceding the argument that has to be formatted.

Format Specifier in printf()

- .precision

.precision	description
.number	For integer specifiers (d, i, o, u, x, X): precision specifies the minimum number of digits to b e written. If the value to be written is shorter than this number, the result is padded with lea ding zeros. The value is not truncated even if the result is longer. A precision of 0 means th at no character is written for the value 0. For a, A, e, E, f and F specifiers: this is the number of digits to be printed after the decimal point (by default, this is 6). For g and G specifiers: This is the maximum number of significant digits to be printed. For s: this is the maximum number of characters to be printed. By default all characters are printed until the ending null character is encountered. If the period is specified without an explicit value for precision, 0 is assumed.
.*	The precision is not specified in the format string, but as an additional integer value argum ent preceding the argument that has to be formatted.

Format Specifier in printf()

- length
  - The *length* sub-specifier modifies the length of the data type.

length	specifiers						
	d i	**u o x X**	**f F e E g G a A**	**c**	**s**	**p**	**n**
(none)	int	unsigned int	double	int	char*	void*	int*
hh	signed char	unsigned char					signed char*
h	short int	unsigned short int					short int*
l	long int	unsigned long int		wint_t	wchar_t*		long int*
ll	long long int	unsigned long long int					long long int*
j	intmax_t	uintmax_t					intmax_t*
z	size_t	size_t					size_t*
t	ptrdiff_t	ptrdiff_t					ptrdiff_t*
L			long double				

Example: printf()

Declarations and Initializations			
char c = 'A', s[] = "Blue moon!";			
Format	**Corresponding Argument**	**How it is printed in its field**	**Remarks**
%c	c	"A"	field width 1 by default
%2c	c	" A"	field width 2, right adjusted
%-3c	c	"A "	field width 3, left adjusted
%s	s	"Blue moon!"	field width 10 by default
%3s	s	"Blue moon!"	more space needed
%.6s	s	"Blue m"	precision 6
%-11.8s	s	"Blue moo "	precision 8, left adjusted

Example: printf()

Declarations and Initializations			
int i = 123; double x = 0.123456789;			
Format	**Corresponding Argument**	**How it is printed in its field**	**Remarks**
%d	i	“123”	field width 3 by default
%05d	i	“ 00123”	padded with zeros
%7o	i	“ 173“	right adjusted, octal
%-9x	i	“7b ”	left adjusted, hexadecimal
%-#9x	i	“0x7b ”	left adjusted, hexadecimal
%10.5f	x	“ 0.12346”	field width 10, precision 5
%-12.5e	x	“1.23457e-01 “	left adjusted, e-format

The Input Function scanf()

- `int scanf ( const char * format, ... );`
- Reads data from **stdin** and stores them according to the parameter *format* into the locations pointed by the additional arguments.
- Format
  - **Whitespace character:** the function will read and ignore any whitespace characters encountered before the next non-whitespace character.
  - **Non-whitespace character, except format specifier (%):** the function will read the next character from the stream
  - **Format specifiers:** It is used to specify the type and format of the data to be retrieved from the stream and stored into the locations pointed by the additional arguments.

Format Specifier in scanf()

- A format specifier for scanf follows this prototype.
  - %[*][width][length]specifier

specifier	Description	Characters extracted
i	Integer	Any number of digits, optionally preceded by a sign (+ or -). Decimal digits assumed by default (0-9), but a 0 prefix introduces octal digit s (0-7), and0x hexadecimal digits (0-f). Signed argument.
d or u	Decimal integer	Any number of decimal digits (0-9), optionally preceded by a sign (+ or -). d is for a signed argument, and u for an unsigned.
o	Octal integer	Any number of octal digits (0-7), optionally preceded by a sign (+ or -). Unsigned argument.
x	Hexadecimal integer	Any number of hexadecimal digits (0-9, a-f, A-F), optionally preceded by 0 x or 0X, and all optionally preceded by a sign (+ or -). Unsigned argument.

Format Specifier in scanf()

specifier	Description	Characters extracted
f, e, g	Floating point number	A series of decimal digits, optionally containing a decimal point, optionally preceeded by a sign (+ or -) and optionally followed by the e or E character and a decimal integer (or some of the other sequences supported by strtod). Implementations complying with C99 also support hexadecimal floating-point format when preceded by 0x or 0X.
c	Character	The next character. If a width other than 1 is specified, the function reads exactly width characters and stores them in the successive locations of the array passed as argument. No null character is appended at the end.
s	String of characters	Any number of non-whitespace characters, stopping at the first whitespace character found. A terminating null character is automatically added at the end of the stored sequence.
p	Pointer address	A sequence of characters representing a pointer. The particular format used depends on the system and library implementation, but it is the same as the one used to format %p in fprintf.

Format Specifier in scanf()

specifier	Description	Characters extracted
[*characters*]	Scanset	Any number of the characters specified between the brackets. A dash (-) that is not the first character may produce non-portable be havior in some library implementations.
[^*characters*]	Negated scanset	Any number of characters none of them specified as *characters* bet ween the brackets.
n	Count	No input is consumed. The number of characters read so far from stdin is stored in the point ed location.
%	%	A % followed by another % matches a single %.

Format Specifier in scanf()

- sub-specifier
  - %**[*][width][length]**specifier

sub-specifier	description
*	An optional starting asterisk indicates that the data is to be read from the stream but igno red (i.e. it is not stored in the location pointed by an argument).
width	Specifies the maximum number of characters to be read in the current reading operation (optional).
length	One of hh, h, l, ll, j, z, t, L (optional). This alters the expected type of the storage pointed by the corresponding argument (see below).

Format Specifier in scanf()

- length

length	specifiers					
	d i	u o x	f e g a	c s [] [^]	p	n
(none)	int*	unsigned int*	float*	char*	void**	int*
hh	signed char*	unsigned char*				signed char*
h	short int*	unsigned short int*				short int*
l	long int*	unsigned long int*	double*	wchar_t*		long int*
ll	long long int*	unsigned long long int*				long long int*
j	intmax_t*	uintmax_t*				intmax_t*
z	size_t*	size_t*				size_t*
t	ptrdiff_t*	ptrdiff_t*				ptrdiff_t*
L			long double*			

Return Value of scanf()

- On success, the function returns the number of items of the argument list successfully filled.
  - This count can match the expected number of items or be less (even zero) due to a matching failure, a reading error, or the reach of the *end-of-file*.
- If a reading error happens or the *end-of-file* is reached while reading, **EOF** is returned.

Example: scanf()

```
[input]
45 , ignore_this  %  C  read_in_this**
```

```
int i;
char c;
char string[15];

scanf("%d , %*s %% %c %5s %5", &i, &c, string, &string[5]);
```

The Functions fprintf() and fscanf()

- The functions fprintf() and fscanf() are file versions of the functions pri ntf() and scanf(), respectively.
  - int fprintf (FILE * fp, const char * format, ...);
  - int fscanf (FILE * fp, const char * format, ...);

- The identifier FILE is defined in stdio.h as a particular structure, with members that describe the current state of a file.
  - The function fprintf() writes to the file pointed by ***fp***.
  - The function fscanf() reads from the file pointed by ***fp***.

The Functions fprintf() and fscanf()

- Three standard file pointers **stdin**, **stdout**, and **stderr** are defined in stdio.h.

Written in C	Name
stdin	standard input file
stdout	standard output file
stderr	standard error file

- fprintf(stdout, ...) is equivalent to printf(...).
- fscanf(stdin, ...) is equivalent to scanf(...).

The Functions sprintf() and sscanf()

- The functions sprintf() and sscanf() are string versions of the printf() and scanf(), respectively.
  - int sprintf (char *s, const char * format, ...);
  - int sscanf (const char *s, const char * format, ...);

```
char str1[] = "1 2 3 go", str2[100] , tmp[100];
int a, b, c;
sscanf(str1, "%d%d%d%s", &a, &b, &c, tmp)i
sprintf(str2, "%s %s %d %d %d\n", tmp, tmp, a, b, c);
printf("%s", str2);
```

```
[output]
go go 1 2 3
```

Files

- Abstractly, a file can be thought of as a stream of characters.
- After a file has been opened, the stream can be accessed with file handling functions in the standard library.
- The file
  - have a name.
  - must be opened before using it.
  - must be closed after using it.
- To prevent accidental misuse, we must tell the system which of the three **activities** we will be performing on it.
  - Activities: *reading, writing, or appending*

The Functions fopen() and fclose()

- FILE * fopen (const char * filename, const char * mode);
- int fclose (FILE * fp);

```
#include <stdio.h>

int main(void)
{
    int a, sum = 13;
    FILE *ifp, *ofp;
    ifp = fopen("my_file", "r"); /* open for reading */
    ofp = fopen("outfile", "w"); /* open for writing */
```

The Functions fopen() and fclose()

```
// read a sequence of integers in the file pointed by ifp
while (fscanf(ifp, "%d", &a) == 1)
    sum += a;

// write the result to the file pointed by ofp
fprintf(ofp, "The sum is %d.\n", sum);

// close files
fclose(ifp);
fclose(ofp);
```

The Functions fopen() and fclose()

- A function call of the form fopen(filename, mode) opens the named file in a particular mode and returns a file pointer.

mode	description
"r"	**read:** Open file for input operations. The file must exist.
"w"	**write:** Create an empty file for output operations. If a file with the same name already exists, its contents are discarded and the file is treated as a new empty file.
"a"	**append:** Open file for output at the end of a file. Output operatio ns always write data at the end of the file, expanding it. The file i s created if it does not exist.

The Functions fopen() and fclose()

- The file will be opened as a text file in default. In order to open a file as a binary file, a "b" character has to be included in the mode string.

mode	description
"rb"	open binary file for “r” mode
"wb"	open binary file for “w” mode
"ab"	open binary file for “a” mode

The Functions fopen() and fclose()

- Each of these modes can end with a **+** character. This means that the file is to be for both reading and writing.

mode	description
"r+"	**read/update:** Open a file for update (both for input and output). The file must exist.
"w+"	**write/update:** Create an empty file and open it for update (both for input and output). If a file with the same name already exists its contents are d iscarded and the file is treated as a new empty file.
"a+"	**append/update:** Open a file for update (both for input and output) with a ll output operations writing data at the end of the file. Repositioning oper ations (fseek, fsetpos, rewind) affects the next input operations, but outp ut operations move the position back to the end of file. The file is create d if it does not exist.

The Example: Double Spacing a File

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

void double_space(FILE *, FILE *);
void prn_info(char *);

int main(int argc, char **argv)
{
    FILE *ifp, *ofp;
    if (argc != 3) {
        prn_info(argv[0]);
        exit(1);
    }
```

```c
    ifp = fopen(argv[1] , "r");
    ofp = fopen(argv[2] , "w");
    double_space(ifp, ofp);
    fclose(ifp);
    fclose(ofp);
    return 0;
}

void double_space(FILE *ifp, FILE *ofp)
{
    int c;
    while ((c = getc(ifp)) != EOF) {
        putc(c, ofp);
        if (c == '\n')
            putc('\n',ofp); // found a newline - duplicate it
    }
}
```

```
void prn_info(char *pgm_name)
{
    printf("\n%s%s%s\n\n%s%s\n\n",
    "Usage: ", pgm_name, " infile outfile“,
    "The contents of infile will be double spaced ",
    "and written to outfile. ");
}
```

- dbl_space   file1   file2
  - The program will read from file1 and write to file2. The contents of file2 will be the same as file1, except that every newline character will have been duplicated.

Accessing a File Randomly

- The library functions fseek() and ftell() are used to access a file randomly.
- **long int ftell (FILE * fp);**
  - Returns the current value of the position indicator.
- **int fseek (FILE * fp, long int offset, int origin);**
  - Sets the position indicator associated with the ***fp*** to a new position (***offset*** bytes from the ***origin***).
  - origin
    - SEEK_SET: Beginning of the file.
    - SEEK_CUR: Current position of the file pointer.
    - SEEK_END: End of the file.

Example: Print a File Backwards

```
/* Print a file backwards. */
#include <stdio.h>

#define MAXSTRING 100

int main(void)
{
    char fname[MAXSTRING];
    int c;
    FILE *fp;
    fprintf(stderr, "\nInput a filename: ") ;
    scanf("%s" , fname);
    ifp = fopen(fname, "rb");
    fseek(ifp, 0, SEEK_END); /* move to end of the file */
    fseek(ifp, -1, SEEK_CUR); /* back up one character */
    while (ftell(ifp) > 0) {
        c = getc(ifp); /* move ahead one character */
        putchar(c);
        fseek(ifp, -2, SEEK_CUR); /* back up two characters */
    }
    return 0;
}
```