

Programming in C

Pradip Dey & Manas Ghosh

CHAPTER 3

Input and Output

OVERVIEW

- Understand what C considers as standard input and output devices.
- Get to know the input and output streams that exist in C to carry out the input and output Tasks.
- Understand that C provides a set of input and output functions.
- Learn the use of single character unformatted input and output functions `getchar()` and `putchar()`.
- Learn to use the formatted input and output functions `scanf()` and `printf()` for .handling multiple input and output.

INTRODUCTION

- Generally ,standard input and output devices are the keyboard and the screen.
- To carry out the input and output, a number of standard functions such as `getchar()`,`putchar()`, `scanf()`, and `printf()` are in-built in C.
- `getchar()` and `putchar()` functions are single- character input and output functions respectively. So, these do not need any formatted inputs or outputs.
- The functions `scanf()` and `printf()` handle multiple variables of all the allowed data types in C. These, therefore, require formatted inputs and outputs.

KEY TERMS

- **Character string** :A chain of characters placed one after another that is dealt as one unit.
- **Control code** :Special characters that specify some positional action on the printing point, also known as cursor.
- **Conversion specifier**: Same as format specifier.
- **Size modifier** :Precedes the conversion code and specifies the kind of data type thereby indicating the number of bytes required for the corresponding variable.
- **White space**: Blank space that causes the input stream to be read up to the next non-white-space character.

KEY TERMS

- **Format specifier :** Identifies the data type, along with width, precision, size and flag, for the respective variables to be outputted to or read in from a standard device.
- **Format string :** A group of characters that contain ordinary character string, conversion code, or control characters arranged in order so that they correspond to the respective control string variables placed next to it in printf() function.
- **Width modifier:** When used in context to the format string, specifies the total number of characters used to display the output or to be read in.

KEY TERMS

- **Precision modifier:** Indicates the number of characters used after the decimal point in the output displayed. The precision option is only used with floats or strings.
- **Flag modifier :** It is a character that specifies one or more of the following:
 - display space if no sign symbol precedes the output
 - inclusion of + or – sign symbol
 - preceding the output Padding the output with leading 0s.
 - the positioning of the output to be displayed
 - Use of alternate form of specifier.

BASIC SCREEN AND KEYBOARD I/O IN C

- C provides several functions which are, in most cases, implemented as routines that call lower-level input/output functions.
- The input and output functions in C are built around the concept of a set of standard data streams being connected from each executing program to the basic input/output devices.
- These standard data streams or files are opened by the operating system and are available to every C and assembler program.
- The input and output functions in C are implemented through a set of standard data streams which connect each executing program to the basic input/output devices.

CONT.

- The input/output functions are of two kinds:
 - non formatted &
 - formatted functions.
- These standard data streams or files are opened by the operating system and are available to every C and assembler program.
- These standard files or streams are called
 - stdin : connected to the keyboard
 - stdout : connected to the screen
 - stderr : connected to the screen
- The following two data streams are also available on MSDOS-based computers.
 - stdaux : connected to the first serial communications port
 - stdprn : connected to the first parallel printer port

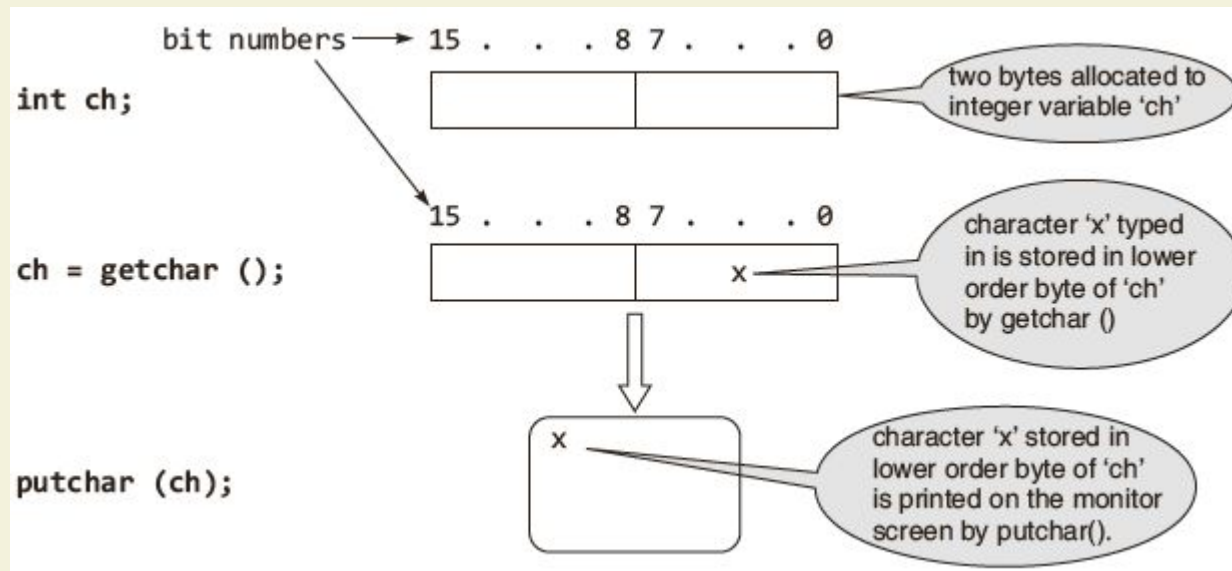
NON-FORMATTED INPUT AND OUTPUT

- Non-formatted input and output can be carried out by standard input-output library functions in C. These can handle one character at a time. For the input functions it does not require **<Enter>** to be pressed after the entry of the character.
- A number of functions provide for character-oriented input and output. The declarations format of two of these are given as follows:
 - `int getchar(void);`(function for character input)
 - `int putchar(int c);`(function of character output)

CONT.

- `getchar()` is an input function that reads a single character from the standard input device, normally a keyboard.
- `putchar()` is an output function that writes a single character on the standard output device, the display screen.
- There are two other functions, `gets()` and `puts()`, that are used to read and write strings from and to the keyboard and the display screen respectively. A string may be defined as an arranged collection of characters.

The following program code displays the character entered through `getchar()` on the screen.



FORMATTED INPUT AND OUTPUT FUNCTIONS

- When input and output is required in a specified format the standard library functions `scanf()` and `printf()` are used.
 - The `scanf()` function allows the user to input data in a specified format. It can accept data of different data types .
 - The `printf()` function allows the user to output data of different data types on the console in a specified format.
- The *format string in `printf()`, enclosed in quotation marks*, has three types of objects:
 - *Ordinary characters: these are copied to output,*
 - *Conversion specifier field: denoted by % containing the codes listed in Table 1. and by optional modifiers such as width, precision, flag, and size,*
 - *Control code: optional control characters such as \n,\b, and \t.*

Table 1 Format specifiers for printf()

Conversion code	Usual variable type	Display
<code>%c</code>	<code>char</code>	single character
<code>%d (%i)</code>	<code>int</code>	signed integer
<code>%e (%E)</code>	<code>float</code> or <code>double</code>	exponential format
<code>%f</code>	<code>float</code> or <code>double</code>	signed decimal
<code>%g (%G)</code>	<code>float</code> or <code>double</code>	use <code>%f</code> or <code>%e</code> , whichever is shorter
<code>%o</code>	<code>int</code>	unsigned octal value
<code>%p</code>	<code>pointer</code>	address stored in pointer
<code>%s</code>	array of <code>char</code>	sequence of characters (string)
<code>%u</code>	<code>int</code>	unsigned decimal integer
<code>%x (%X)</code>	<code>int</code>	unsigned hex value
<code>%%</code>	<code>none</code>	no corresponding argument is converted, prints only a <code>%</code>
<code>%n</code>	<code>pointer to int</code>	the corresponding argument is a pointer to an integer into which the number of characters displayed is placed.

FLAG CHARACTER USED IN PRINTF()

flag	Meaning
-	Left justify the display
+	Display positive or negative sign of value
space	Display space if there is no sign
0	Pad with leading zeros
#	Use alternate form of specifier

Here are a couple of examples using the flag options.

```
printf("number=%06.1f\n", 5.5); printf("%-+6.1f=number\n", 5.5);
```

The output of these two statements in order is:

number =

0	0	0	5	.	5
---	---	---	---	---	---

 ,

+	5	.	5		
---	---	---	---	--	--

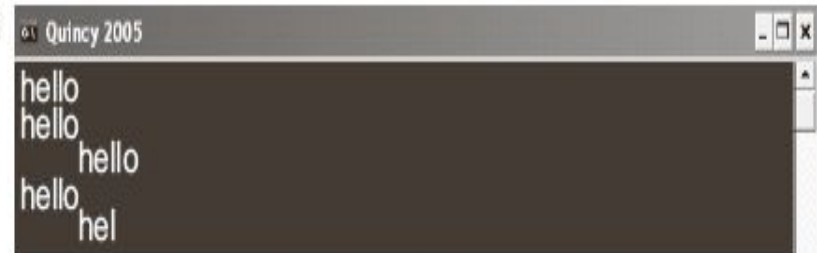
 = number

EXAMPLE:

```
#include <stdio.h>
int main()
{
    printf("%s", "hello");          /* first printf() */
    printf("\n%3s", "hello");      /* second printf() */
    printf("\n%10s", "hello");     /* third printf() */
    printf("\n%-10s", "hello");    /* fourth printf() */
    printf("\n%10.3s", "hello");   /* fifth printf() */
    return 0;
}
```

The output for the respective printf() functions would be as follows:

h e l l o	←	output from first printf()
h e l l o	←	output from second printf()
h e l l o	←	output from third printf()
h e l l o	←	output from fourth printf()
h e l	←	output from fifth printf()



```
hello
hello
hello
hello
hel
```


EXAMPLE:

```
printf("number=%3d\n", 10);  
printf("number=%2d\n", 10);  
printf("number=%1d\n", 10);  
printf("number=%7.2f\n", 5.4321);  
printf("number=%.2f\n", 5.4391);  
printf("number=%.9f\n", 5.4321);  
printf("number=%f\n", 5.4321);
```

The output of these five statements in order are:

number =

	1	0
--	---	---

number =

1	0
---	---

number =

1	0
---	---

number =

			5	.	4	3
--	--	--	---	---	---	---

number =

5	.	4	4
---	---	---	---

number =

5	.	4	3	2	1	0	0	0	0	0
---	---	---	---	---	---	---	---	---	---	---

number =

5	.	4	3	2	1	0	0
---	---	---	---	---	---	---	---

Output screen snapshot:



```
Quincy 2005  
number= 10  
number=10  
number=10  
number= 5.43  
number=5.44  
number=5.432100000  
number=5.432100
```

USES OF # FLAG WITH FORMAT SPECIFIER

Among the flags, the only complexity is in the use of the # modifier. What this modifier does depends on the type of format specifier, that is, the conversion code it is used with.

flag with format specifier	Action
%#O	Adds a leading 0 to the octal number printed
%#x or X	Adds a leading 0x or 0X to the hex number printed
%#f or e	Ensures that the decimal point is printed
%#g or G	Displays trailing zeros in g or G type conversion and ensures decimal point is printed in floating-point number, even though it is a whole number

SIZE MODIFIER

- The effects of the size modifiers that transform the conversion code are shown in this Table below.
- Examples of the use of size are as follows:
- %hd /* short integer */
- %ld /* long integer */
- %Lf /* long double */

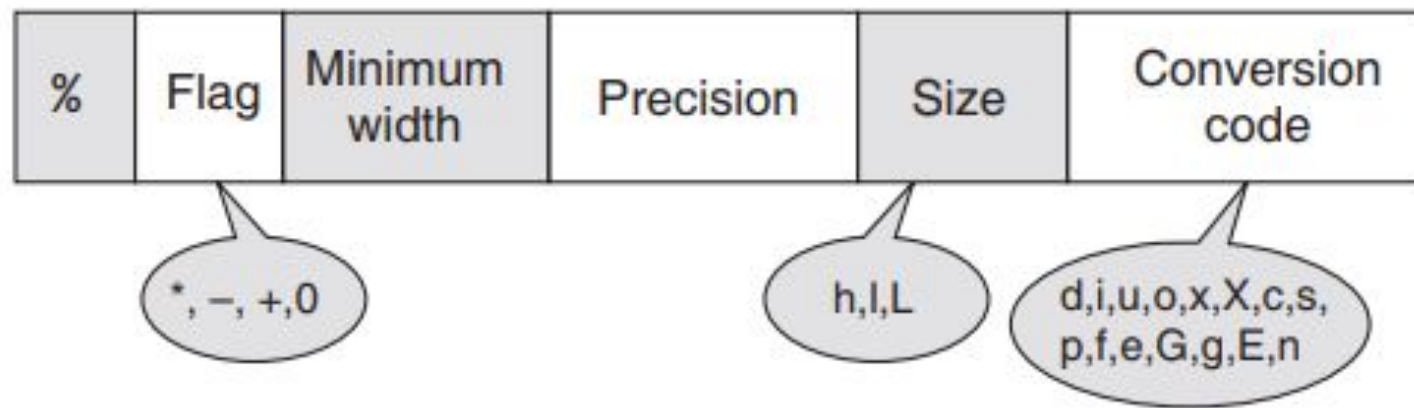
Size modifier	Conversion code	Converts to
l	d i o u x	long int
h	d i o u x	short int
l	e f	double
L	e f	long double

COMMONLY USED CONTROL CODES

- There are the control codes also known as escape sequences.
- If any of these are included in the format string, the corresponding ASCII control code is sent to the screen, or output device, which should produce the effect listed.
- The conversion specifier field is used to format printed values, often to arrange things nicely in columns.
- The control code and conversion specifier may be embedded within the character string.

Table : List of commonly used control codes

Control code	Action
\b	Backspace
\f	Form feed
\n	New line
\r	Carriage return
\t	Horizontal tab
\'	Single quote
\0	Null



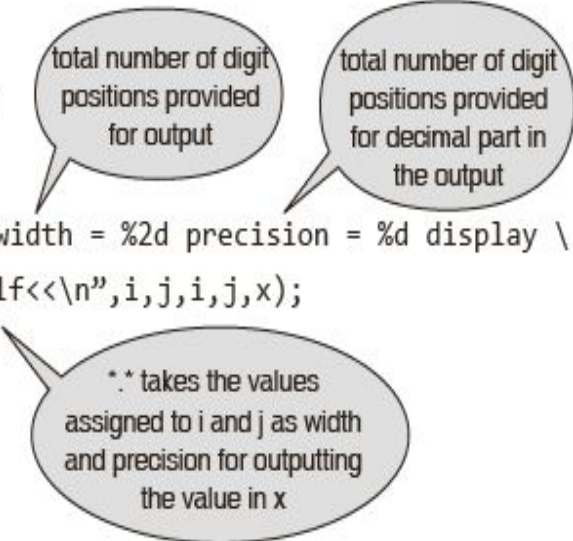
%<flag(s)><width><precision><size>conversioncode

RUNTIME ADJUSTMENT AND PRECISION IN PRINTF()

- The correct way to adjust field *width and precision* at run time is to replace the *width and/or precision* with a star (*).
- Include appropriate integer variables in the parameter list.
- The values of these integer variables representing width and precision will be used before the actual variable to be converted is taken from the parameter list.
- The next slide is showing a program that describe the feature of ***Runtime adjustment and precision in printf()***.

EXAMPLE : RUNTIME ADJUSTMENT AND PRECISION IN PRINTF()

```
#include <stdio.h>
int main()
{
    double x=1234.567890;
    int i=8,j=2;
    while(i<12)
    {
        j=2;
        while(j<5)
        {
            printf("width = %2d precision = %d display \
                >>*. *lf<<\n",i,j,i,j,x);
        }
        j++;
    }
    return 0;
}
```



The program displays the effects of various widths and precisions for output of a double variable. Here is the output.

```
width =  8 precision = 2 display >> 1234.57<<
width =  8 precision = 3 display >>1234.568<<
width =  8 precision = 4 display >>1234.5679<<
width =  9 precision = 2 display >> 1234.57<<
width =  9 precision = 3 display >> 1234.568<<
width =  9 precision = 4 display >>1234.5679<<
width = 10 precision = 2 display >> 1234.57<<
width = 10 precision = 3 display >> 1234.568<<
width = 10 precision = 4 display >> 1234.5679<<
width = 11 precision = 2 display >> 1234.57<<
width = 11 precision = 3 display >> 1234.568<<
width = 11 precision = 4 display >> 1234.5679<<
```

The >> and << symbols are used to indicate the limits of the output field. Note that the variables i and j appear twice in the parameter list, the first time to give the values in the annotation and the second time to actually control the output.

INPUT FUNCTION SCANF()

- The scanf() function works in much the same way as the printf(). It has the general form :
scanf("control_string",variable1_address,variable2_address,...);
- In scanf(), the control string or format string, that consists of a list of format specifiers, indicates the format and type of data to be read in from the standard input device, which is the keyboard, for storing in the corresponding address of variables specified.
- There must be the same number of format specifiers and addresses as there are input fields.
- scanf() returns the number of input fields successfully scanned, converted, and stored.
- If scanf() attempts to read end-of-file, the return value is EOF. If no fields were stored, the return value is 0.

CONT.

- **White space:** This causes the input stream to be read up to the next non-white-space character.
- **Ordinary character string:** Anything except white space or % characters. The next character in the input stream must match this character.
- **Conversion specifier field:** This is a % character, followed by an optional * character, which suppresses the conversion, followed by an optional non-zero decimal integer specifying the maximum field width, an optional h, l, or L to control the length of the conversion, and finally a non-optional conversion specifier.
- Important is that use of h, l, or L will affect the type of pointer which must be used.

FORMAT SPECIFIERS FOR SCANF()

- The format string in scanf() has the following general form:
“< character string >< % conversion specifier field >”
- Each ‘conversion specifier field’ is coded as follows:
%[*]<width><size><conversion-code>
- Table:

Format Specifiers for scanf()

Conversion code	Usual variable type	Action
%c	char	Read a single character
%d(%i)	int	Read a signed decimal integer
%e(%E)	float or double	Read signed decimal
%f	float or double	Read signed decimal
%g(%G)	float or double	Read signed decimal
%o	int	Read octal value
%p	pointer	Read in hex address stored in pointer
%s	array of char	Read sequence of characters (string)
%u	int	Read unsigned decimal integer

Conversion code	Usual variable type	Action
%x(%X)	int	Read unsigned hex value
%%	none	A single % character in the input stream is expected. There is no corresponding argument.
%n	pointer to int	No characters in the input stream are matched. The corresponding argument is a pointer to an integer into which the number of characters read is placed.
[...]	array of char	Read a string of matching characters

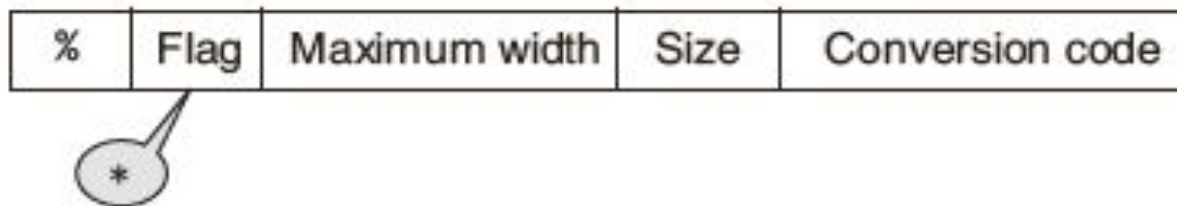
CONT.

- Each *conversion (or format) specifier* begins with the per cent character, %, after which come the following, in the given order.
 1. An optional assignment-suppression character, *, which states that the value being read will not be assigned to an argument, but will be dropped.
 2. An optional *width specifier*, <width>, which *designates* the maximum number of characters to be read that compose the value for the associated argument. Encountering white space, before the entire width is scanned, terminates the input of this value and moves to the next.

CONT.

3. An optional conversion-code modifier, <size>, which modifies the conversion code to accept format for a type of :
- h = short int,
 - l = long int, if the format specifiers provide for an integer conversion,
 - l = double, if the format specifiers provide for a floating-point conversion, and
 - L = long double, which is valid only with floating point conversions. The format specifiers in scanf() are shown in Fig.

Fig. Parts of conversion specifier field for scanf()



SCANF() VS PRINTF()

- The scanf() function returns the number of variables successfully read in.
- 2. The printf() function returns a number that is equal to the number of characters printed.