9

C Formatted Input/Output

All the news that's fit to print.

—Adolph S. Ochs

What mad pursuit? What struggle to escape?

—John Keats

Remove not the landmark on the boundary of the fields.

—Amenemope

OBJECTIVES

In this chapter you will learn:

- To use input and output streams.
- To use all print formatting capabilities.
- To use all input formatting capabilities.
- To print with field widths and precisions.
- To use formatting flags in the `printf` format control string.
- To output literals and escape sequences.
- To format input using `scanf`.

Outline

9.1 Introduction

9.2 Streams

9.3 Formatting Output with `printf`

9.4 Printing Integers

9.5 Printing Floating-Point Numbers

9.6 Printing Strings and Characters

9.7 Other Conversion Specifiers

9.8 Printing with Field Widths and Precision

9.9 Using Flags in the `printf` Format Control String

9.10 Printing Literals and Escape Sequences

9.11 Reading Formatted Input with `scanf`

9.1 Introduction

- **In this chapter**
  - **Presentation of results**
  - **`scanf` and `printf`**
  - **Streams (input and output)**
    - **`gets`, `puts`, `getchar`, `putchar` (in `<stdio.h>`)**

9.2 Streams

- **Streams**
  - **Sequences of characters organized into lines**
    - **Each line consists of zero or more characters and ends with newline character**
    - **ANSI C must support lines of at least 254 characters**
  - **Performs all input and output**
  - **Can often be redirected**
    - **Standard input – keyboard**
    - **Standard output – screen**
    - **Standard error – screen**
    - **More in Chapter 11**

9.3 Formatting Output with `printf`

- `printf`
  - Precise output formatting
    - Conversion specifications: flags, field widths, precisions, etc.
  - Can perform rounding, aligning columns, right/left justification, inserting literal characters, exponential format, hexadecimal format, and fixed width and precision
- Format
  - `printf(` *format-control-string, other-arguments* `);`
  - Format control string: describes output format
  - Other-arguments: correspond to each conversion specification in format-control-string
    - Each specification begins with a percent sign(%), ends with conversion specifier

Common Programming Error 9.1

Forgetting to enclose a format-control-string in quotation marks is a syntax error.

Good Programming Practice 9.1

Format outputs neatly for presentation to make program outputs more readable and reduce user errors.

9.4 Printing Integers

- **Integer**
  - **Whole number (no decimal point): `25`, `0`, `-9`**
  - **Positive, negative, or zero**
  - **Only minus sign prints by default (later we will change this)**

Conversion specifier	Description
`d`	Display as a signed decimal integer.
`i`	Display as a signed decimal integer. [*Note:* The `i` and `d` specifiers are different when used with `scanf`.]
`o`	Display as an unsigned octal integer.
`u`	Display as an unsigned decimal integer.
`x` or `X`	Display as an unsigned hexadecimal integer. `X` causes the digits `0-9` and the letters `A-F` to be displayed and `x` causes the digits `0-9` and `a-f` to be displayed.
`h` or `l` (letter `l`)	Place before any integer conversion specifier to indicate that a `short` or `long` integer is displayed, respectively. Letters `h` and `l` are more precisely called **length modifiers**.

Fig. 9.1 | Integer conversion specifiers.

Outline

fig09_02.c

(1 of 2)

```
1  /* Fig 9.2: fig09_02.c */
2  /* Using the integer conversion specifiers */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     printf( "%d\n", 455 );
8     printf( "%i\n", 455 );  /* i same as d in printf */
9     printf( "%d\n", +455 );
10    printf( "%d\n", -455 );
11    printf( "%hd\n", 32000 );
12    printf( "%ld\n", 2000000000L ); /* L suffix makes literal a long */
13    printf( "%o\n", 455 );
14    printf( "%u\n", 455 );
```

d and **i** specify signed integers

h specifies a **short** number

l specifies a **long** number

o specifies an octal integer

u specifies an unsigned integer

```
15    printf( "%u\n", -455 );
16    printf( "%x\n", 455 );
17    printf( "%X\n", 455 );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

x and **X** specify hexadecimal integers

```
455
455
455
-455
32000
2000000000
707
455
4294966841
1c7
1C7
```

Outline

fig09_02.c

(2 of 2)

Common Programming Error 9.2

Printing a negative value with a conversion specifier that expects an `unsigned` value.

9.5 Printing Floating-Point Numbers

- **Floating Point Numbers**
  - **Have a decimal point (`33.5`)**
  - **Exponential notation (computer's version of scientific notation)**
    - **`150.3` is `1.503 x 10²` in scientific**
    - **`150.3` is `1.503E+02` in exponential**

Conversion specifier	Description
`e` or `E`	Display a floating-point value in exponential notation.
`f`	Display floating-point values in fixed-point notation.
`g` or `G`	Display a floating-point value in either the floating-point form `f` or the exponential form `e` (or `E`), based on the magnitude of the value.
`L`	Place before any floating-point conversion specifier to indicate that a `long double` floating-point value is displayed.

Fig. 9.3 | Floating-point conversion specifiers.

Error-Prevention Tip 9.1

When outputting data, be sure that the user is aware of situations in which data may be imprecise due to formatting (e.g., rounding errors from specifying precisions).

Outline

fig09_04.c

```
1  /* Fig 9.4: fig09_04.c */
2  /* Printing floating-point numbers with
3     floating-point conversion specifiers */
4
5  #include <stdio.h>
6
7  int main( void )
8  {
9     printf( "%e\n", 1234567.89 );
10    printf( "%e\n", +1234567.89 );
11    printf( "%e\n", -1234567.89 );
12    printf( "%E\n", 1234567.89 );
13    printf( "%f\n", 1234567.89 );
14    printf( "%g\n", 1234567.89 );
15    printf( "%G\n", 1234567.89 );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

e and **E** specify exponential notation

f specifies fixed-point notation

g and **G** specify either exponential or fixed-point notation depending on the number's size

```
1.234568e+006
1.234568e+006
-1.234568e+006
1.234568E+006
1234567.890000
1.23457e+006
1.23457E+006
```

9.6 Printing Strings and Characters

- `c`
  - Prints `char` argument
  - Cannot be used to print the first character of a string
- `s`
  - Requires a pointer to `char` as an argument
  - Prints characters until `NULL` (`'\0'`) encountered
  - Cannot print a `char` argument
- **Remember**
  - Single quotes for character constants (`'z'`)
  - Double quotes for strings `"z"` (which actually contains two characters, `'z'` and `'\0'`)

Common Programming Error 9.3

Using `%c` to print a string is an error. The conversion specifier `%c` expects a `char` argument. A string is a pointer to `char` (i.e., a `char *`).

Common Programming Error 9.4

Using `%s` to print a `char` argument, on some systems, causes a fatal execution-time error called an access violation. The conversion specifier `%s` expects an argument of type pointer to `char`.

Common Programming Error 9.5

Using single quotes around character strings is a syntax error. Character strings must be enclosed in double quotes.

Common Programming Error 9.6

Using double quotes around a character constant creates a pointer to a string consisting of two characters, the second of which is the terminating null. A character constant is a single character enclosed in single quotes.

Outline

fig09_05.c

```
1  /* Fig 9.5: fig09_05c */
2  /* Printing strings and characters */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char character = 'A'; /* initialize char */
8     char string[] = "This is a string"; /* initialize char array */
9     const char *stringPtr = "This is also a string"; /* char pointer */
10
11    printf( "%c\n", character );
12    printf( "%s\n", "This is a string" );
13    printf( "%s\n", string );
14    printf( "%s\n", stringPtr );
15
16    return 0; /* indicates successful termination */
17
18 } /* end main */
```

c specifies a character will be printed

s specifies a string will be printed

```
A
This is a string
This is a string
This is also a string
```

9.7 Other Conversion Specifiers

- p
  - Displays pointer value (address)
- n
  - Stores number of characters already output by current `printf` statement
  - Takes a pointer to an integer as an argument
  - Nothing printed by a `%n` specification
  - Every `printf` call returns a value
    - Number of characters output
    - Negative number if error occurs
- %
  - Prints a percent sign
  - %%

Portability Tip 9.1

The conversion specifier p displays an address in an implementation-defined manner (on many systems, hexadecimal notation is used rather than decimal notation).

Common Programming Error 9.7

Trying to print a literal percent character using % rather than %% in the format control string. When % appears in a format control string, it must be followed by a conversion specifier.

Conversion specifier	Description
p	Display a pointer value in an implementation-defined manner.
n	Store the number of characters already output in the current `printf` statement. A pointer to an integer is supplied as the corresponding argument. Nothing is displayed.
%	Display the percent character.

Fig. 9.6 | Other conversion specifiers.

Outline

fig09_07.c

(1 of 2)

```
1  /* Fig 9.7: fig09_07.c */
2  /* Using the p, n, and % conversion specifiers */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int *ptr;      /* define pointer to int */
8     int x = 12345; /* initialize int x */
9     int y;         /* define int y */
10
11    ptr = &x;      /* assign address of x to ptr */
12    printf( "The value of ptr is %p\n", ptr );
13    printf( "The address of x is %p\n\n", &x );
14
15    printf( "Total characters printed on this line:%n", &y );
16    printf( " %d\n\n", y );
17
18    y = printf( "This line has 28 characters\n%n" );
19    printf( "%d characters were printed\n\n", y );
20
21    printf( "Printing a %% in a format control string\n" );
```

p specifies a memory address will be printed

n stores the number of characters printed on a line

% prints a percent sign

```
22
23    return 0; /* indicates successful termination */
24
25 } /* end main */
```

```
The value of ptr is 0012FF78
The address of x is 0012FF78

Total characters printed on this line: 38

This line has 28 characters
28 characters were printed

Printing a % in a format control string
```

Outline

fig09_07.c

(2 of 2)

9.8 Printing with Field Widths and Precision

- **Field width**
  - **Size of field in which data is printed**
  - **If width larger than data, default right justified**
    - **If field width too small, increases to fit data**
    - **Minus sign uses one character position in field**
  - **Integer width inserted between % and conversion specifier**
  - **%4d – field width of 4**

Outline

fig09_08.c

(1 of 2)

```
1  /* Fig 9.8: fig09_08.c */
2  /* Printing integers right-justified */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     printf( "%4d\n", 1 );
8     printf( "%4d\n", 12 );
9     printf( "%4d\n", 123 );
10    printf( "%4d\n", 1234 );
11    printf( "%4d\n\n", 12345 );
12
13    printf( "%4d\n", -1 );
14    printf( "%4d\n", -12 );
```

A field width of 4 will make C attempt to print the number in a 4-character space

Note that C considers the minus sign a character

```
15    printf( "%4d\n", -123 );
16    printf( "%4d\n", -1234 );
17    printf( "%4d\n", -12345 );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

```
   1
  12
 123
1234
12345

  -1
 -12
-123
-1234
-12345
```

Outline

fig09_08.c

(2 of 2)

Common Programming Error 9.8

Not providing a sufficiently large field width to handle a value to be printed can offset other data being printed and can produce confusing outputs. Know your data!

9.8 Printing with Field Widths and Precision

- **Precision**
  - **Meaning varies depending on data type**
  - **Integers (default `1`)**
    - **Minimum number of digits to print**

      If data too small, prefixed with zeros
  - **Floating point**
    - **Number of digits to appear after decimal (`e` and `f`)**

      For `g` – maximum number of significant digits
  - **Strings**
    - **Maximum number of characters to be written from string**
  - **Format**
    - **Use a dot (`.`) then precision number after `%`**

      ```
      %.3f
      ```

Outline

fig09_09.c

(1 of 2)

```
1  /* Fig 9.9: fig09_09.c */
2  /* Using precision while printing integers,
3     floating-point numbers, and strings */
4  #include <stdio.h>
5
6  int main( void )
7  {
8     int i = 873;                  /* initialize int i */
9     double f = 123.94536;         /* initialize double f */
10    char s[] = "Happy Birthday"; /* initialize char array s */
11
12    printf( "Using precision for integers\n" );
13    printf( "\t%.4d\n\t%.9d\n\n", i, i );
14
15    printf( "Using precision for floating-point numbers\n" );
16    printf( "\t%.3f\n\t%.3e\n\t%.3g\n\n", f, f, f );
```

Precision for integers specifies the minimum number of characters to be printed

Precision for the **g** specifier controls the maximum number of significant digits printed

Precision for **f** and **e** specifiers controls the number of digits after the decimal point

```
17
18    printf( "Using precision for strings\n" );
19    printf( "\t%.11s\n", s );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

Precision for strings specifies the maximum number of characters to be printed

```
Using precision for integers
        0873
        000000873

Using precision for floating-point numbers
        123.945
        1.239e+002
        124

Using precision for strings
        Happy Birth
```

Outline

fig09_09.c

(2 of 2)

9.9 Using Flags in the `printf` Format Control String

- **Flags**
  - **Supplement formatting capabilities**
  - **Place flag immediately to the right of percent sign**
  - **Several flags may be combined**

Flag	Description
`-` (minus sign)	Left justify the output within the specified field.
`+` (plus sign)	Display a plus sign preceding positive values and a minus sign preceding negative values.
space	Print a space before a positive value not printed with the `+` flag.
#	Prefix `0` to the output value when used with the octal conversion specifier `o`.
	Prefix `0x` or `0X` to the output value when used with the hexadecimal conversion specifiers `x` or `X`.
	Force a decimal point for a floating-point number printed with `e`, `E`, `f`, `g` or `G` that does not contain a fractional part. (Normally the decimal point is printed only if a digit follows it.) For `g` and `G` specifiers, trailing zeros are not eliminated.
`0` (zero)	Pad a field with leading zeros.

Fig. 9.10 | Format control string flags.

Outline

fig09_11.c

```
1  /* Fig 9.11: fig09_11.c */
2  /* Right justifying and left justifying values */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     printf( "%10s%10d%10c%10f\n\n", "hello", 7, 'a', 1.23 );
8     printf( "%-10s%-10d%-10c%-10f\n", "hello", 7, 'a', 1.23 );
9
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

- flag left justifies characters in a field

```
     hello         7         a  1.230000

hello     7         a         1.230000
```

Outline

fig09_12.c

```
1  /* Fig 9.12: fig09_12.c */
2  /* Printing numbers with and without the + flag */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     printf( "%d\n%d\n", 786, -786 );
8     printf( "%+d\n%+d\n", 786, -786 );
9
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

+ flag forces a plus sign on positive numbers

```
786
-786
+786
-786
```

Outline

fig09_13.c

```
1  /* Fig 9.13: fig09_13.c */
2  /* Printing a space before signed values
3     not preceded by + or - */
4  #include <stdio.h>
5
6  int main( void )
7  {
8     printf( "% d\n% d\n", 547, -547 );
9
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

Space flag forces a space on positive numbers

```
 547
-547
```

Outline

fig09_14.c

```
1  /* Fig 9.14: fig09_14.c */
2  /* Using the # flag with conversion specifiers
3     o, x, X and any floating-point specifier */
4  #include <stdio.h>
5
6  int main( void )
7  {
8     int c = 1427;      /* initialize c */
9     double p = 1427.0; /* initialize p */
10
11    printf( "%#o\n", c );
12    printf( "%#x\n", c );
13    printf( "%#X\n", c );
14    printf( "\n%g\n", p );
15    printf( "%#g\n", p );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

flag prefixes a **0** before octal integers

flag prefixes a **0x** before hexadecimal integers

flag forces a decimal point on floating-point numbers with no fractional part

```
02623
0x593
0X593

1427
1427.00
```

Outline

fig09_15.c

```
1  /* Fig 9.15: fig09_15.c */
2  /* Printing with the 0( zero ) flag fills in leading zeros */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     printf( "%+09d\n", 452 );
8     printf( "%09d\n", 452 );
9
10    return 0; /* indicates successful termination */
11
12 } /* end main */
```

0 flag fills empty spaces with zeros

```
+00000452
000000452
```

9.10 Printing Literals and Escape Sequences

- **Printing Literals**
  - **Most characters can be printed**
  - **Certain "problem" characters, such as the quotation mark "**
  - **Must be represented by escape sequences**
    - **Represented by a backslash \ followed by an escape character**

Common Programming Error 9.9

Attempting to print as literal data in a `printf` statement a single quote, double quote or backslash character without preceding that character with a backslash to form a proper escape sequence is an error.

Escape sequence	Description
`\'` (single quote)	Output the single quote (') character.
`\"` (double quote)	Output the double quote (") character.
`\?` (question mark)	Output the question mark (?) character.
`\\` (backslash)	Output the backslash (\) character.
`\a` (alert or bell)	Cause an audible (bell) or visual alert.
`\b` (backspace)	Move the cursor back one position on the current line.
`\f` (new page or form feed)	Move the cursor to the start of the next logical page.
`\n` (newline)	Move the cursor to the beginning of the next line.
`\r` (carriage return)	Move the cursor to the beginning of the current line.
`\t` (horizontal tab)	Move the cursor to the next horizontal tab position.
`\v` (vertical tab)	Move the cursor to the next vertical tab position.

Fig. 9.16 | Escape sequences.

9.11 Formatting Input with `scanf`

- `scanf`
  - Input can be formatted much like output can
  - `scanf` conversion specifiers are slightly different from those used with `printf`

Good Programming Practice 9.2

When inputting data, prompt the user for one data item or a few data items at a time. Avoid asking the user to enter many data items in response to a single prompt.

Good Programming Practice 9.3

Always consider what the user and your program will do when (not if) incorrect data is entered—for example, a value for an integer that is nonsensical in a program's context, or a string with missing punctuation or spaces.

Conversion specifier	Description
Integers	
`d`	Read an optionally signed decimal integer. The corresponding argument is a pointer to an `int`.
`i`	Read an optionally signed decimal, octal or hexadecimal integer. The corresponding argument is a pointer to an `int`.
`o`	Read an octal integer. The corresponding argument is a pointer to an unsigned `int`.
`u`	Read an unsigned decimal integer. The corresponding argument is a pointer to an unsigned `int`.
`x` or `X`	Read a hexadecimal integer. The corresponding argument is a pointer to an unsigned `int`.
`h` or `l`	Place before any of the integer conversion specifiers to indicate that a `short` or `long` integer is to be input.

Fig. 9.17 | Conversion specifiers for **`scanf`**. (Part 1 of 3.)

Conversion specifier	Description
Floating-point numbers	
`e`, `E`, `f`, `g` or `G`	Read a floating-point value. The corresponding argument is a pointer to a floating-point variable.
`l` or `L`	Place before any of the floating-point conversion specifiers to indicate that a `double` or `long double` value is to be input. The corresponding argument is a pointer to a `double` or `long double` variable.
Characters and strings	
`c`	Read a character. The corresponding argument is a pointer to a `char`; no null `('\0')` is added.
`s`	Read a string. The corresponding argument is a pointer to an array of type `char` that is large enough to hold the string and a terminating null `('\0')` character—which is automatically added.

Fig. 9.17 | Conversion specifiers for **`scanf`**. (Part 2 of 3.)

Conversion specifier	Description
Scan set	
[*scan characters*]	Scan a string for a set of characters that are stored in an array.
Miscellaneous	
`p`	Read an address of the same form produced when an address is output with `%p` in a `printf` statement.
`n`	Store the number of characters input so far in this call to `scanf`. The corresponding argument is a pointer to an `int`.
`%`	Skip a percent sign `(%)` in the input.

Fig. 9.17 | Conversion specifiers for **`scanf`**. (Part 3 of 3.)

Outline

fig09_18.c

```
1  /* Fig 9.18: fig09_18.c */
2  /* Reading integers */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int a;
8     int b;
9     int c;
10    int d;
11    int e;
12    int f;
13    int g;
14
15    printf( "Enter seven integers: " );
16    scanf( "%d%i%i%i%o%u%x", &a, &b, &c, &d, &e, &f, &g );
17
18    printf( "The input displayed as decimal integers is:\n" );
19    printf( "%d %d %d %d %d %d %d\n", a, b, c, d, e, f, g );
20
21    return 0; /* indicates successful termination */
22
23 } /* end main */
```

d specifies a decimal integer will be input

i specifies an integer will be input

o specifies an octal integer will be input

u specifies an unsigned decimal integer will be input

x specifies a hexadecimal integer will be input

```
Enter seven integers: -70 -70 070 0x70 70 70 70
The input displayed as decimal integers is:
-70 -70 56 112 56 70 112
```

Outline

fig09_19.c

```
1  /* Fig 9.19: fig09_19.c */
2  /* Reading floating-point numbers */
3  #include <stdio.h>
4
5  /* function main begins program execution */
6  int main( void )
7  {
8     double a;
9     double b;
10    double c;
11
12    printf( "Enter three floating-point numbers: \n" );
13    scanf( "%le%lf%lg", &a, &b, &c );
14
15    printf( "Here are the numbers entered in plain\n" );
16    printf( "floating-point notation:\n" );
17    printf( "%f\n%f\n%f\n", a, b, c );
18
19    return 0; /* indicates successful termination */
20
21 } /* end main */
```

e, **f**, and **g** specify a floating-point number will be input

l specifies a **double** or **long double** will be input

```
Enter three floating-point numbers:
1.27987 1.27987e+03 3.38476e-06
Here are the numbers entered in plain
floating-point notation:
1.279870
1279.870000
0.000003
```

Outline

fig09_20.c

```
1  /* Fig 9.20: fig09_20.c */
2  /* Reading characters and strings */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char x;
8     char y[ 9 ];
9
10    printf( "Enter a string: " );
11    scanf( "%c%s", &x, y );
12
13    printf( "The input was:\n" );
14    printf( "the character \"%c\" ", x );
15    printf( "and the string \"%s\"\n", y );
16
17    return 0; /* indicates successful termination */
18
19 } /* end main */
```

c specifies a character will be input

s specifies a string will be input

```
Enter a string: Sunday
The input was:
the character "S" and the string "unday"
```

Outline

fig09_21.c

```c
/* Fig 9.21: fig09_21.c */
/* Using a scan set */
#include <stdio.h>

/* function main begins program execution */
int main( void )
{
   char z[ 9 ]; /* define array z */

   printf( "Enter string: " );
   scanf( "%[aeiou]", z ); /* search for set of characters */

   printf( "The input was \"%s\"\n", z );

   return 0; /* indicates successful termination */

} /* end main */
```

[] specifies only the initial segment of a string that contains the characters in brackets will be read

```
Enter string: ooeeooahah
The input was "ooeeooa"
```

Outline

fig09_22.c

```
1  /* Fig 9.22: fig09_22.c */
2  /* Using an inverted scan set */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     char z[ 9 ];
8
9     printf( "Enter a string: " );
10    scanf( "%[^aeiou]", z ); /* inverted scan set */
11
12    printf( "The input was \"%s\"\n", z );
13
14    return 0; /* indicates successful termination */
15
16 } /* end main */
```

[] and **^** specify only the initial segment of a string that does **not** contain the characters in brackets will be read

```
Enter a string: String
The input was "Str"
```

Outline

fig09_23.c

```
1  /* Fig 9.23: fig09_23.c */
2  /* inputting data with a field width */
3  #include <stdio.h>
4
5  int main( void )
6  {
7     int x;
8     int y;
9
10    printf( "Enter a six digit integer: " );
11    scanf( "%2d%d", &x, &y );
12
13    printf( "The integers input were %d and %d\n", x, y );
14
15    return 0; /* indicates successful termination */
16
17 } /* end main */
```

A field width of 2 tells C to only read the first 2 characters of that input

```
Enter a six digit integer: 123456
The integers input were 12 and 3456
```

Outline

fig09_24.c

```c
/* Fig 9.24: fig09_24.c */
/* Reading and discarding characters from the input stream */
#include <stdio.h>

int main( void )
{
   int month1;
   int day1;
   int year1;
   int month2;
   int day2;
   int year2;

   printf( "Enter a date in the form mm-dd-yyyy: " );
   scanf( "%d%*c%d%*c%d", &month1, &day1, &year1 );

   printf( "month = %d  day = %d  year = %d\n\n", month1, day1, year1 );

   printf( "Enter a date in the form mm/dd/yyyy: " );
   scanf( "%d%*c%d%*c%d", &month2, &day2, &year2 );

   printf( "month = %d  day = %d  year = %d\n", month2, day2, year2 );

   return 0; /* indicates successful termination */

} /* end main */
```

`*` is a wildcard—**`scanf`** will disregard anything between the two inputs on either side of it

```
Enter a date in the form mm-dd-yyyy: 11-18-2003
month = 11  day = 18  year = 2003

Enter a date in the form mm/dd/yyyy: 11/18/2003
month = 11  day = 18  year = 2003
```