%d : Scan an integer as a signed [decimal](http://en.wikipedia.org/wiki/Decimal) number.

%i : Scan an integer as a signed number. Similar to %d, but interprets the number as [hexadecimal](http://en.wikipedia.org/wiki/Hexadecimal) when preceded by 0x and [octal](http://en.wikipedia.org/wiki/Octal) when preceded by 0. For example, the string 031 would be read as 31 using %d, and 25 using %i. The flag h in %hi indicates conversion to a short and hh conversion to a char.

%u : Scan for decimal unsigned int (Note that in the C99 standard the input value minus sign is optional, so if a negative[[*clarification needed*](http://en.wikipedia.org/wiki/Wikipedia:Please_clarify)] number is read, no errors will arise and the result will be the [two's complement](http://en.wikipedia.org/wiki/Two%27s_complement), effectively ignoring the negative sign in most cases. See [strtoul](http://en.wikipedia.org/wiki/Strtoul)().[[*not in citation given*](http://en.wikipedia.org/wiki/Wikipedia:Verifiability)]) Correspondingly, %hu scans for an unsigned short and %hhu for an unsigned char.

%f : Scan a [floating-point](http://en.wikipedia.org/wiki/Floating-point) number in normal ([fixed-point](http://en.wikipedia.org/wiki/Fixed-point_arithmetic)) notation.

%g, %G : Scan a floating-point number in either normal or exponential notation. %g uses lower-case letters and %G uses upper-case.

%x, %X : Scan an integer as an unsigned [hexadecimal](http://en.wikipedia.org/wiki/Hexadecimal) number.

%o : Scan an integer as an [octal](http://en.wikipedia.org/wiki/Octal) number.

%s : Scan a [character string](http://en.wikipedia.org/wiki/Character_string). The scan terminates at [whitespace](http://en.wikipedia.org/wiki/Whitespace_%28computer_science%29). A [null character](http://en.wikipedia.org/wiki/Null_character) is stored at the end of the string, which means that the buffer supplied must be at least one character longer than the specified input length.

%c : Scan a character (char). No [null character](http://en.wikipedia.org/wiki/Null_character) is added.

[whitespace](http://en.wikipedia.org/wiki/Whitespace_%28computer_science%29): Any whitespace characters trigger a scan for zero or more [whitespace](http://en.wikipedia.org/wiki/Whitespace_%28computer_science%29) characters. The number and type of whitespace characters do not need to match in either direction.

%lf : Scan as a [double](http://en.wikipedia.org/wiki/Double_precision_floating-point_format) floating-point number.

%Lf : Scan as a [long double](http://en.wikipedia.org/wiki/Long_double) floating-point number.

The above can be used in compound with numeric modifiers and the l, L modifiers which stand for "long" in between the percent symbol and the letter. There can also be numeric values between the percent symbol and the letters, preceding the long modifiers if any, that specifies the number of characters to be scanned. An optional [asterisk](http://en.wikipedia.org/wiki/Asterisk) (\*) right after the percent symbol denotes that the datum read by this format specifier is not to be stored in a variable. No argument behind the format string should be included for this dropped variable.

The ff modifier in printf is not present in scanf, causing differences between modes of input and output. The ll and hh modifiers are not present in the C90 standard, but are present in the C99 standard.[[1]](http://en.wikipedia.org/wiki/Scanf_format_string#cite_note-1)

An example of a format string is

"%7d%s %c%lf"

The above format string scans the first seven characters as a decimal integer, then reads the remaining as a string until a space, new line or tab is found, then scans the first non-whitespace character following and a double-precision floating-point number afterwards.

Error handling

scanf is usually used in situations when the program cannot guarantee that the input is in the expected format. Therefore a robust program must check whether the scanf call succeeded and take appropriate action. If the input was not in the correct format, the erroneous data will still be on the input stream and must be read and discarded before new input can be read. An alternative method of reading input, which avoids this, is to use [fgets](http://en.wikipedia.org/wiki/Fgets) and then examine the string read in. The last step can be done by [sscanf](http://en.wikipedia.org/wiki/Sscanf), for example.

|  |  |  |
| --- | --- | --- |
| Type | Use | Example |
| d | Signed decimal integer (int8\_t, int16\_t) | -7 |
| u | Unsigned decimal integer (uint8\_t, uint16\_t) | 7 |
| f | Floating point number (float) | 7.4 |
| c | Character (char) | n |
| x or X | Hexadecimal integer | beef or BEEF |
| o | Octal | 777 |
| e or E | Scientific notation | 6.022e23 or 2.99E9 |

|  |  |
| --- | --- |
| Flag | Description |
| # | Forces "alternate form" of conversion. For %x and %X, it puts 0x or 0X before the hex output. It does not affect most of the other types. |
| 0 | Pads the conversion on the left with zeros (instead of spaces as usual) |
| - | The conversion is left-aligned within the display width (instead of right-aligned as usual) |
| space | Puts a space in front of signed conversions (%d) if the number is positive (where a - sign would appear for a negative number). |
| + | Forces the conversion to display a sign (+ or -) |

|  |  |  |  |
| --- | --- | --- | --- |
| printf, scanf Conversion Letters and Matching Types | | | |
| **Letter** | **Type of Matching Argument** | **Example** | **Output** |
| % | *none* ( [See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "%%" ); | % |
| d, i | int ( [See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm#%22)) | printf( "%i", 17 ); | 17 |
| u | unsigned int (*Converts to decimal*) | printf( "%u", 17u ); | 17 |
| o | unsigned int (*Converts to octal*) | printf( "%o", 17 ); | 21 |
| x | unsigned int (*Converts to lower-case hex*) | printf( "%x", 26 ); | 1a |
| X | unsigned int (*Converts to upper-case hex*) | printf( "%X", 26 ); | 1A |
| f, F | double ( [See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "%f", 3.14 ); | 3.140000 |
| e, E | double ( [See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "%e", 31.4 ); | 3.140000e+01 |
| g, G | double ( [See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "%g, %g", 3.14, 0.0000314 ); | 3.14, 3.14e-05 |
| a, A | double ( [See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "%a", 31.0 ); | 0x1.fp+0 |
| c | int ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "%c", 65 ); | A |
| s | *string* ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "%s", "Hello" ); | Hello |
| p | void\* ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | int a = 1; printf( "%p", &a ); | 0064FE00 |
| n | int\* ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | int a; printf( "ABC%n", &a ); | ABC  (*a==3*) |

|  |  |  |  |
| --- | --- | --- | --- |
| **Format Control** | **Description** | **Example** | **Output** |
| **flags** | The flag characters may appear in any order and have the following meanings: | | |
| -      left-justify within the field.  ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "|%3i|%-3i|", 12, 12); | |·12|12·| |
| +      Forces positive numbers to include a leading plus sign. | printf( "%+i", 17); | +17 |
| *space* Forces positive number to include a leading space.  ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "|% i|", 12); | |·12| |
| #      This flag forces the output to be in some *alternate form*.  ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "%#X", 26); | 0X1A |
| 0      Pad with zeros rather than spaces.  ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | printf( "|%04i|", 12); | |0012| |
| **minimum field-width** | After converting any value to a string, the field width represents the minimum number of characters in the resulting string.  ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm).)  If the converted value has fewer characters, then the resulting string is *padded* with spaces (or zeros) on the left (or right) by default (or if the appropriate flag is used.) | printf( "|%5s|", "ABC"); | |··ABC| |
| Sometimes the minimum field width isn't known at compile-time, and must be computed at run-time.  (For example, printing a table where the width of a column depends on the widest column value in the input.)  In this case the field width can be specified as an asterisk ("\*"), which acts like a *place-holder* for an int value used for the field width.  The value appears in the argument list before the expression being converted. | printf( "|%-\*s|", 5, "ABC" ); | |ABC··| |
| **.precision** | A period by itself implies a precision of zero.  A precision may be replaced with an asterisk ("\*"), which works exactly the same as for an asterisk minimum field width described above.  The meaning of a precision depends on the type of conversion done.  Only the conversions listed below are defined: | | |
| When used with floating-point conversion letters (a, A, e, E, f, F, g, and G) the precision specifies how many digits will appear to the right of the decimal point.  The default precision is six.  (For conversion letters g and G, the precision is actually the maximum number of significant digits.)  The value displayed is always rounded, but note this doesn't change the matching expression in any way.  If the precision is zero, no decimal point appears at all (but see "#" flag above). | printf( "|%5.2f|", 3.147 );  printf( "|%5.2G|", 3.147 ); | |·3.15|  |··3.1| |
| When used with integer conversion letters (d, i, o, u, x, and X) the precision specifies the minimum number of digits to appear.  Leading zeros are added as needed. | printf( "|%6.4i|", 17 ); | |··0017| |
| When used with string conversions (letter "s") the precision specifies the maximum number of bytes written.  ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm).)  If the string is too long it will be truncated. | printf( "|%-5.3s|", "ABCD" ); | |ABC··| |
| **length** | A *length modifier* is used to exactly specify the type of the matching argument.  Since most types are promoted to int or double a length modifier is rarely used.  However it is used for long and other types that don't have an explicit conversion letter of their own.  Note that specific length modifiers only make sense in combination with specific conversion letters.  Using undefined combinations causes unpredictable results.  The length modifiers and their meanings are: | | |
| |  |  | | --- | --- | | hh | Legal for integer conversions (d, i, o, u, x, or X conversion letters), specifies the matching argument is a signed char for d and i, and unsigned char for o, u, x, and X.  For an n conversion letter the matching argument is a pointer to a signed char.  (New as of C99.) | | printf( "%hhi", 300 ); ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | 44 |
| |  |  | | --- | --- | | h | The same as hh, except h specifies the argument is a short or unsigned short. | | printf( "%hi", 300 ); | 300 |
| |  |  | | --- | --- | | l | (This is the letter *ell* and not the digit *one*.)  The same as hh, except l specifies the argument is a long or unsigned long.  For a c conversion the l length modifier specifies a matching type of wint\_t, and for s the matching type is wchar\_t.  ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm).) | | long a = 300, b = (long) 1.0E+14; printf( "%li\n%i", a, b );    printf( "%lc:%ls", L'A', L"ABC" ); | 300 276447232   A:ABC |
| |  |  | | --- | --- | | ll | (This is the letters *ell-ell* and not the digits *one-one*.)  The same as hh, except ll specifies the argument is a long long or unsigned long long.  (New as of C99.) | | printf( "%#llX", 300 ); | 0X12C |
| |  |  | | --- | --- | | j | The same as hh, except j specifies the argument is a intmax\_t or uintmax\_t.  (These types and length modifier are new as of C99.) | | printf( "%ji", 17 ); | 17 |
| |  |  | | --- | --- | | z | The same as hh, except z specifies the argument is a size\_t.  (This length modifier is new as of C99.) | | printf( "%zi", sizeof(int) ); | 4 |
| |  |  | | --- | --- | | t | The same as hh, except t specifies the argument is a ptrdiff\_t.  (This length modifier is new as of C99.) | | char a[5] = "abcd"; printf( "%ti", &(a[3]) - &(a[1]) ); | 2 |
| |  |  | | --- | --- | | L | Legal for floating point conversions (a, A, e, E, f, F, g, and G conversion letters), specifies the matching argument is a long double. | | printf( "%Lf", 3.14L ); | 3.140000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Letter** | **Type of Matching Argument** | **Auto-skip Leading White-Space** | **Example** | **Sample Matching Input** |
| **%** | % (a literal, matched but not converted or assigned) | **no** | int anInt; scanf("%i%%", &anInt); | 23% |
| **d** | int ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **yes** | int anInt; long l; scanf("%d %ld", &anInt, &l); | -23 200 |
| **i** | int ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **yes** | int anInt; scanf("%i", &anInt); | 0x23 |
| **o** | unsigned int ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **yes** | unsigned int aUInt; scanf("%o", &aUInt); | 023 |
| **u** | unsigned int ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **yes** | unsigned int aUInt; scanf("%u", &aUInt); | 23 |
| **x** | unsigned int ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **yes** | unsigned int aUInt; scanf("%d", &aUInt); | 1A |
| **a, e, f, g** | float or double ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **yes** | float f; double d; scanf("%f %lf", &f, &d); | 1.2 3.4 |
| **c** | char ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **no** | char ch; scanf(" %c", &ch); | Q |
| **s** | array of char ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **yes** | char s[30]; scanf("%29s", s); | hello |
| **p** | void ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **yes** | int\* pi; void\* ptr; scanf("%p", &ptr); pi = (int\*) ptr; | 0064FE00 |
| **n** | int ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **no** | int x, cnt; scanf("X: %d%n", &x, &cnt); | X: 123  (*cnt==6*) |
| **[** | array of char ([See note](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm)) | **no** | char s1[64], s2[64]; scanf(" %[^\n]", s1); scanf("%[^\t] %[^\t]", s1, s2); | Hello World field1   field2 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Conversion Modifier** | **Description** | **Example** | **Matching Input** | **Results** |
| **\*** | Assignment Supression.  This modifier causes the corresponding input to be matched and converted, but not assigned (no matching argument is needed). | int anInt; scanf("%\*s %i", &anInt); | Age:·29 | anInt==29, return value==1 |
| **maximum field-width** | This is the maximum number of character to read from the input.  Any remaining input is left unread.  (**Always** use this with "%s" and "%[...]" in **all** production quality code!  (No exceptions!)  You should use one less than the size of the array used to hold the result.) | int anInt; char s[10]; scanf("%2i", &anInt);  scanf("%9s", s); | 2345  VeryLongString | anInt==23, return value==1  s=="VeryLongS"  return value==1 |
| **length modifier** | This specifies the exact type of the matching arugment.  These length codes are the same as the [printf length modifiers](file:///D:\Downloads\Reference\cp%20quickinfo\C%20Printf%20and%20Scanf%20Reference.htm#printfLen), except as noted below: | | | |
| |  |  | | --- | --- | | l | The same for integer conversions (d, i, o, u, x, or X conversion letters), specifies the matching argument is a long int.  For floating point (a, A, e, E, f, F, g, or G conversion letters) an "l" has no effect in a printf, **but with scanf, an ell specifies a double**. | | double d; scanf("%lf", &d); | 3.14 | d==3.14 return value==1 |

char buf[BUFSIZ]; /\* Buffer for a line of input. \*/

int age;

fprintf( stderr, "Please enter your age: " );

while ( fgets( buf, sizeof(buf), stdin ) != NULL )

{

if ( sscanf( buf, "%i", &age ) != 1 )

break;

// Do some sort of error processing:

fprintf( stderr, "\nError reading in the age, please try again.\n" );

}