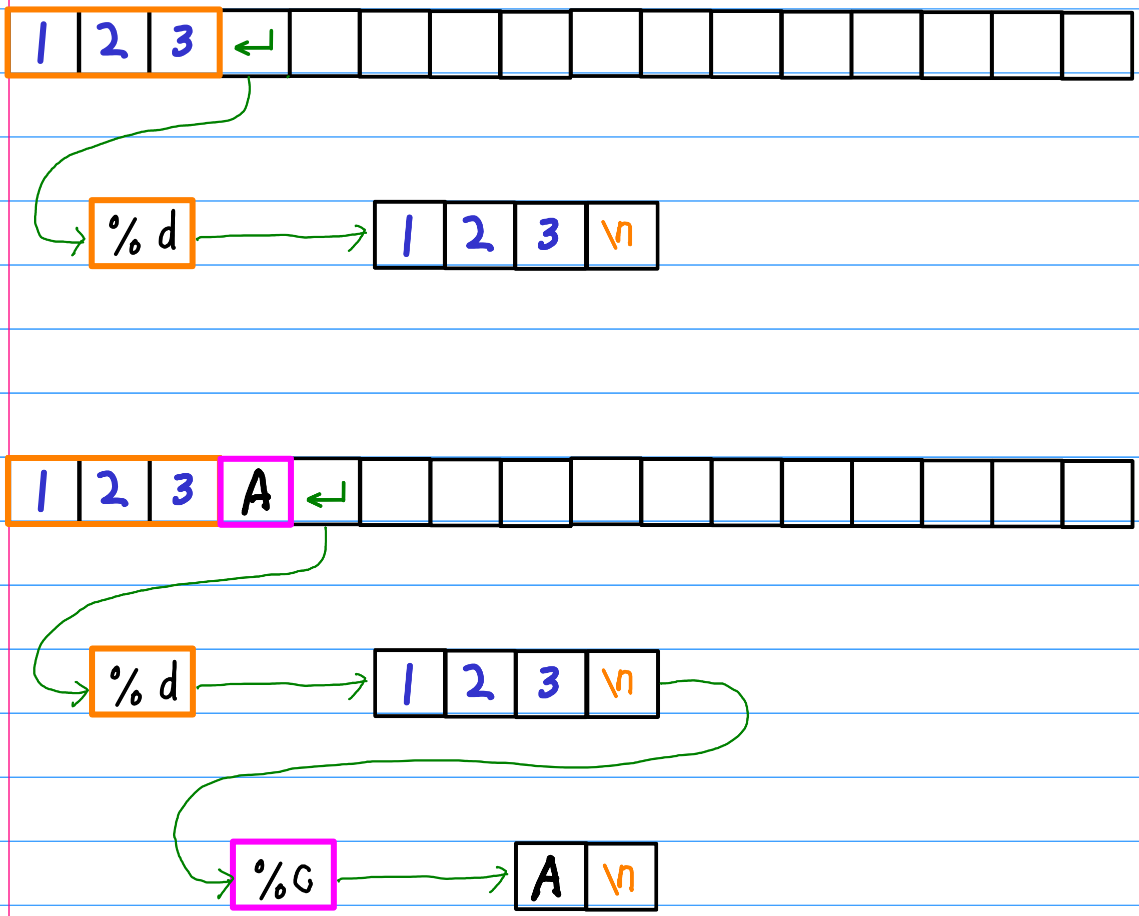
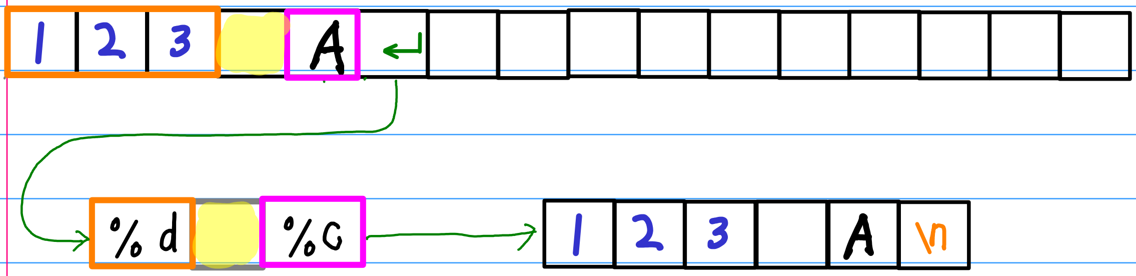
C Programming Day16.B

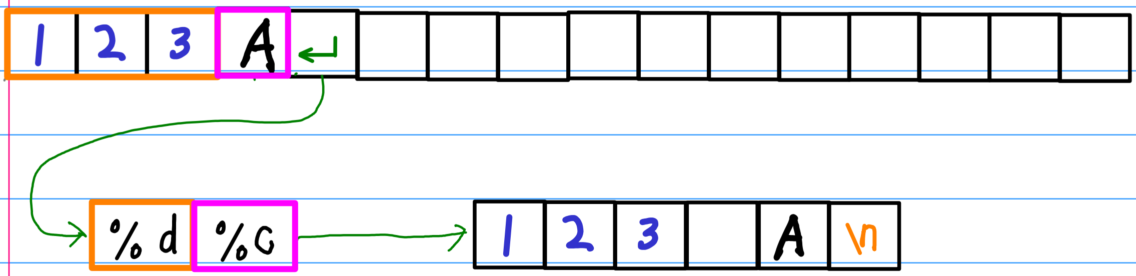
2017.11.14

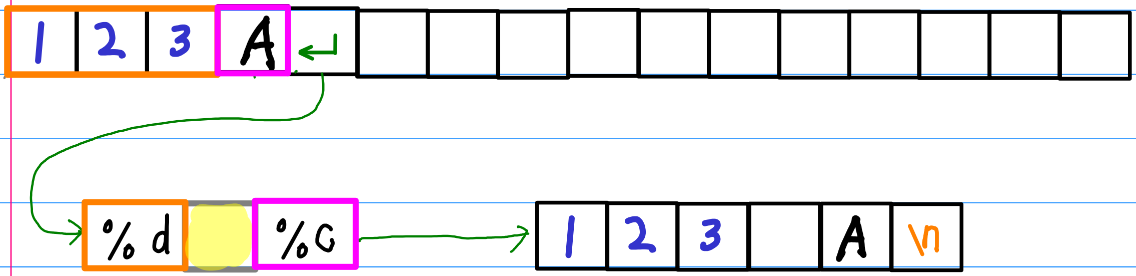
printf(), scanf()
formated io

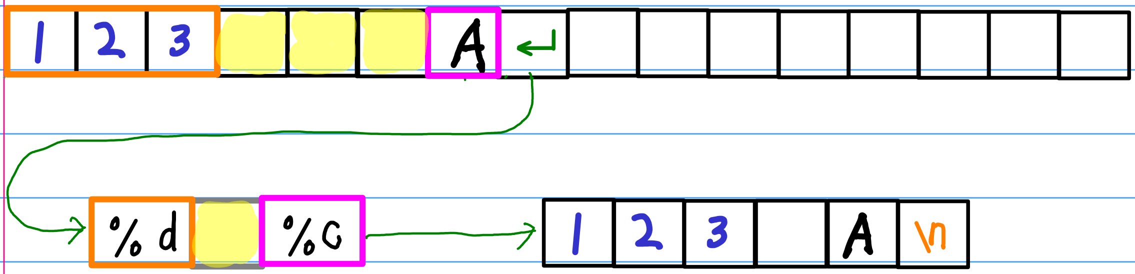


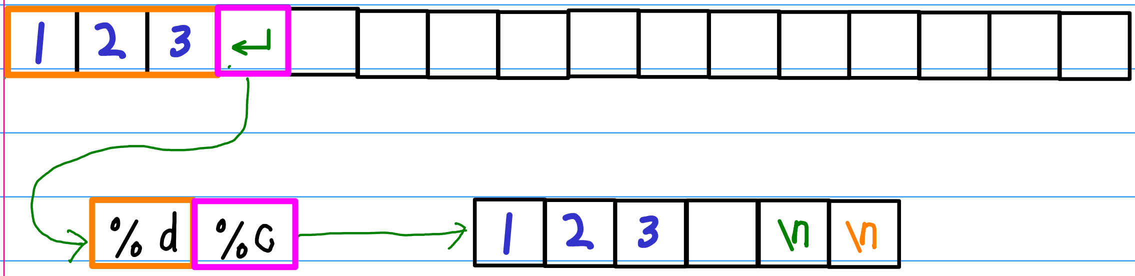
1
2
3
↵
%d
1
2
3
\n
1
2
3
A
↵
%d
1
2
3
\n
%c
A
\n

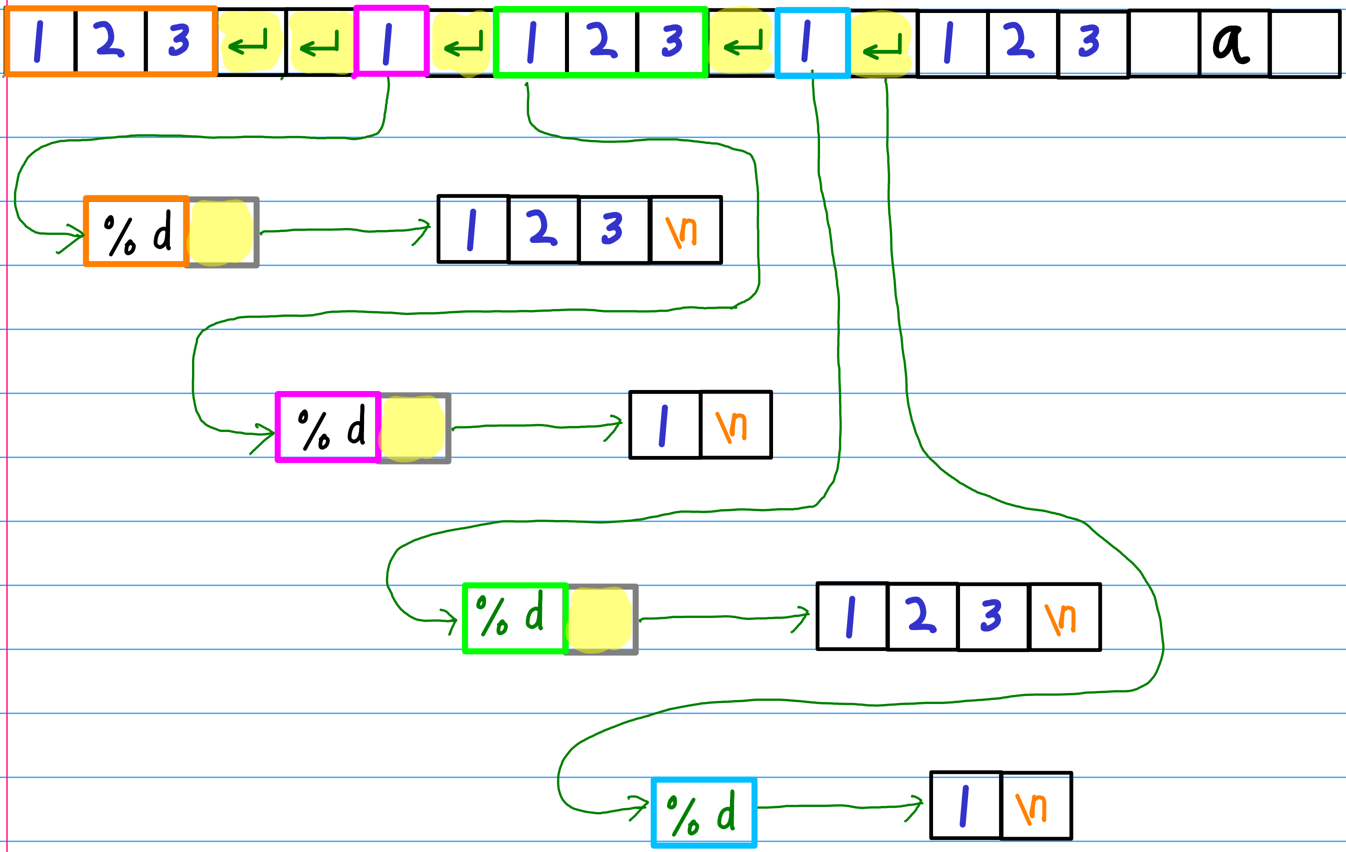

1 2 3 A ↵
% d % c
1 2 3 A \n

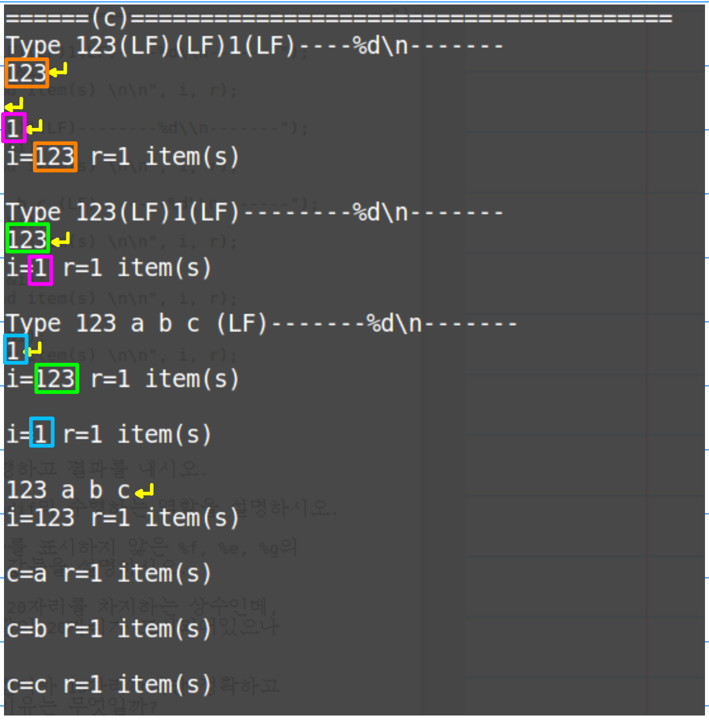

1 2 3 A ↵
% d % c
1 2 3 A \n


1 2 3 A ↵
% d % c
1 2 3 A \n

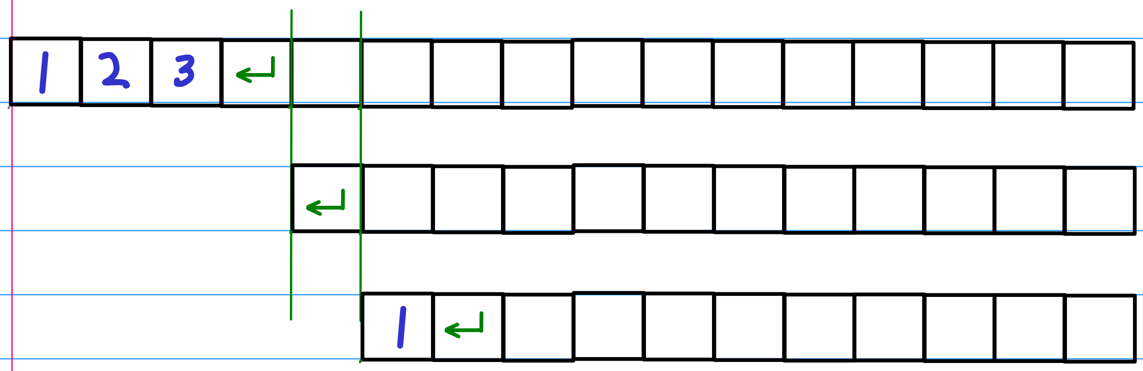

1 2 3 A ↵
% d % c
1 2 3 A \n

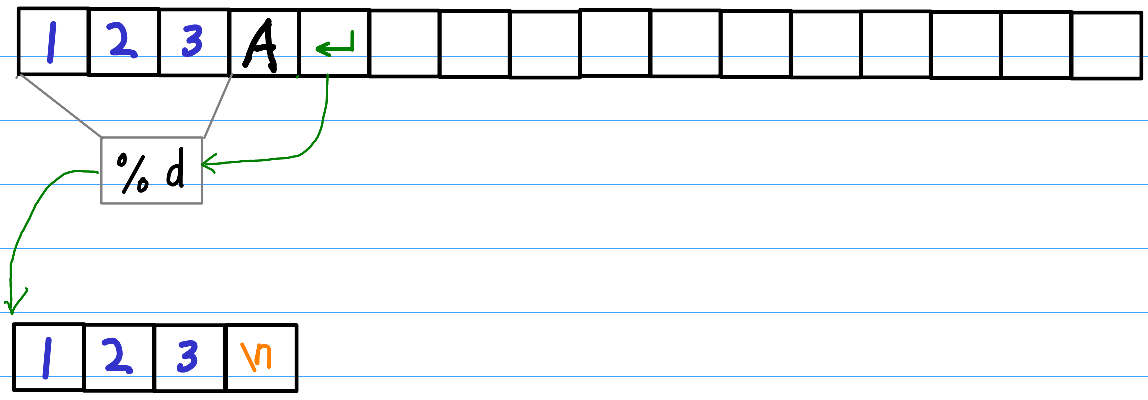

1 2 3 ↵
% d % c
1 2 3 \n \n

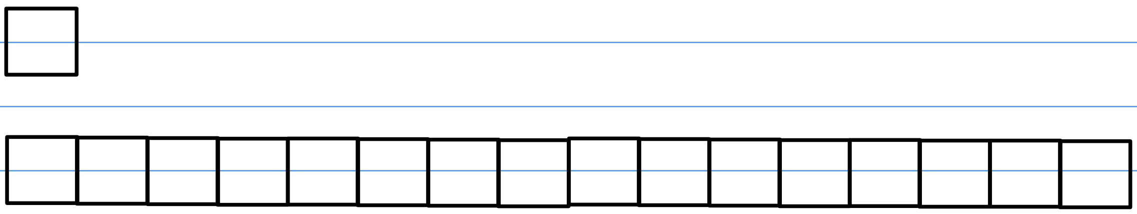




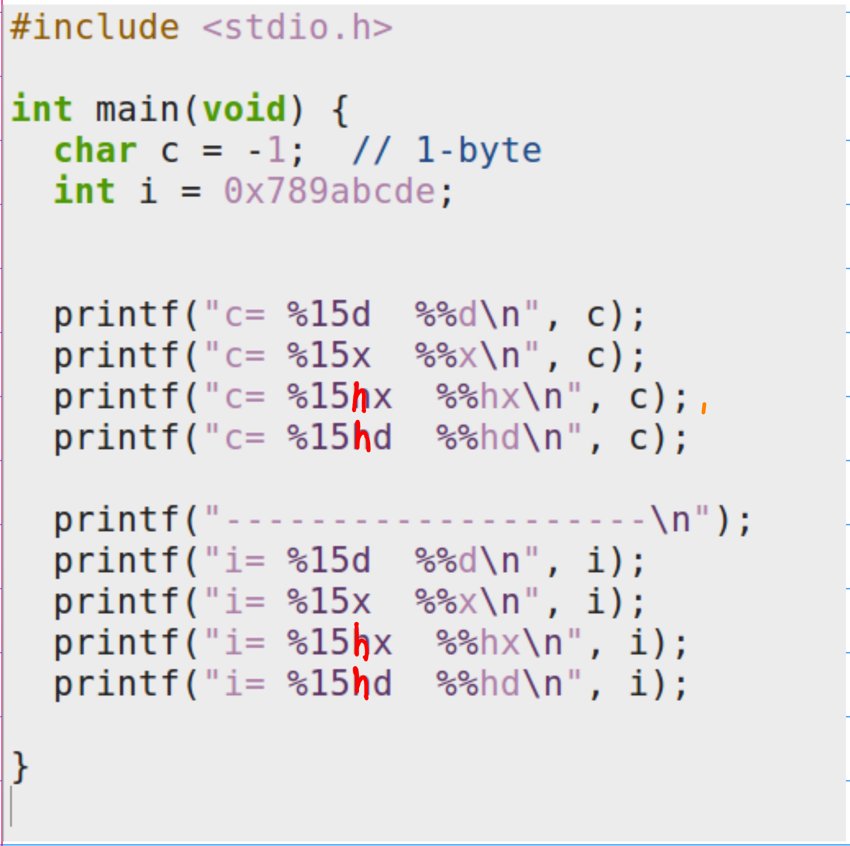
```
======(c)==========================================
Type 123(LF)(LF)1(LF)----%d\n-------
123↵
↵
1↵
i=123 r=1 item(s)

Type 123(LF)1(LF)--------%d\n-------
123↵
i=1 r=1 item(s)

Type 123 a b c (LF)-------%d\n-------
1↵
i=123 r=1 item(s)

i=1 r=1 item(s)

123 a b c↵
i=123 r=1 item(s)

c=a r=1 item(s)

c=b r=1 item(s)

c=c r=1 item(s)
```


1
2
3
↵
↵
1
↵


1
2
3
A
↵
%d
1
2
3
\n

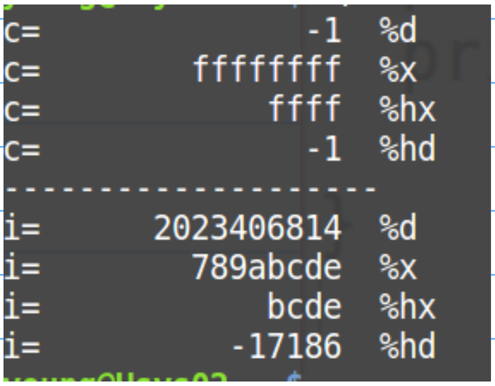


%hx   %hd

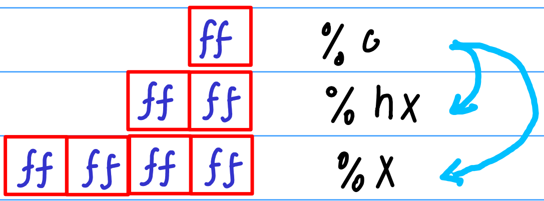


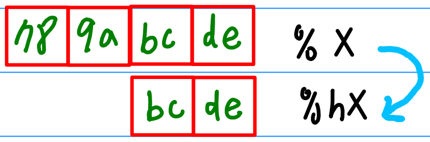
```c
#include <stdio.h>

int main(void) {
  char c = -1;  // 1-byte
  int i = 0x789abcde;


  printf("c= %15d  %%d\n", c);
  printf("c= %15x  %%x\n", c);
  printf("c= %15hx  %%hx\n", c);
  printf("c= %15hd  %%hd\n", c);

  printf("-------------------\n");
  printf("i= %15d  %%d\n", i);
  printf("i= %15x  %%x\n", i);
  printf("i= %15hx  %%hx\n", i);
  printf("i= %15hd  %%hd\n", i);

}
```

short
⇒ 2-byte integer
   4 - hexa digits

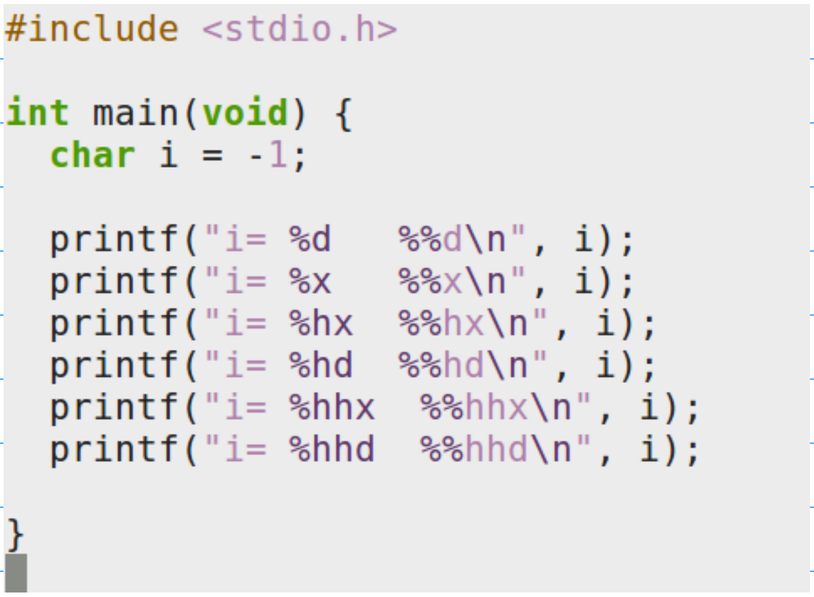


```
c=              -1  %d
c=        ffffffff  %x
c=            ffff  %hx
c=              -1  %hd
-------------------
i=      2023406814  %d
i=        789abcde  %x
i=            bcde  %hx
i=          -17186  %hd
```

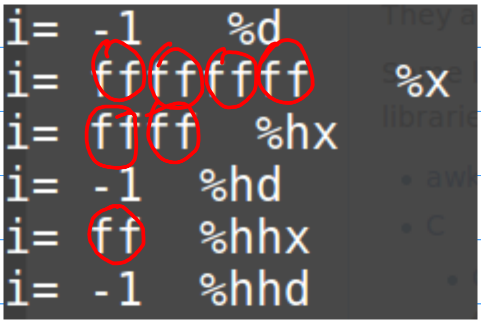




```
-17186     b     c     d     e
           1011  1100  1101  1110
           0100  0011  0010  0001
                                1
           ----------------------
           0100  0011  0010  0010
           4     3     2     2
           ⇒ 17186
```



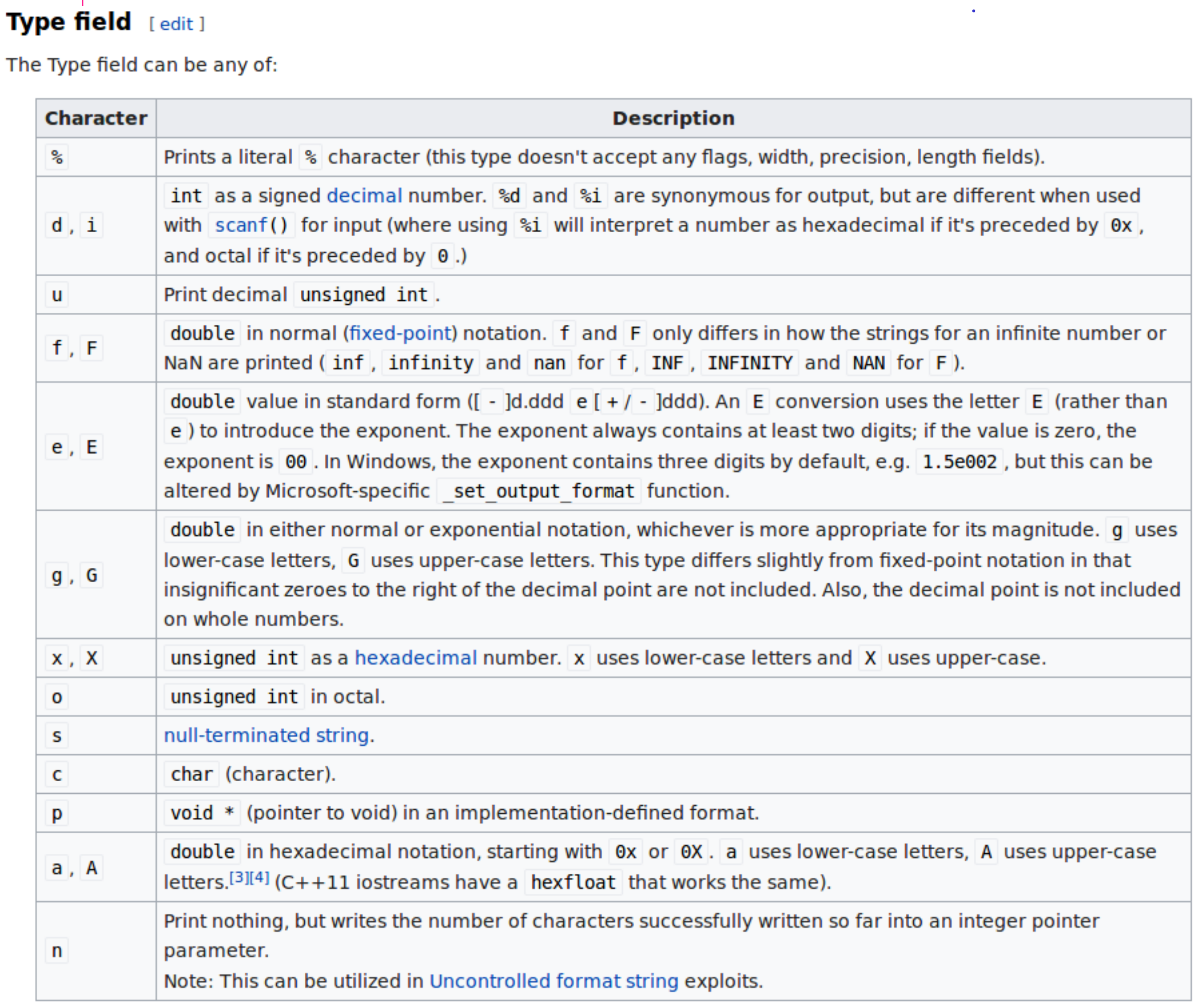
```c
#include <stdio.h>

int main(void) {
  char i = -1;

  printf("i= %d   %%d\n", i);
  printf("i= %x   %%x\n", i);
  printf("i= %hx  %%hx\n", i);
  printf("i= %hd  %%hd\n", i);
  printf("i= %hhx  %%hhx\n", i);
  printf("i= %hhd  %%hhd\n", i);

}
```



```
i= -1    %d
i= ffffffff    %x
i= ffff   %hx
i= -1   %hd
i= ff   %hhx
i= -1   %hhd
```

https://en.wikipedia.org/

$$\begin{tikzcd} \% + & e \arrow[d] & f \arrow[dr] & & g \\ & \text{exponential} & & \text{fixed pt} & \end{tikzcd}$$

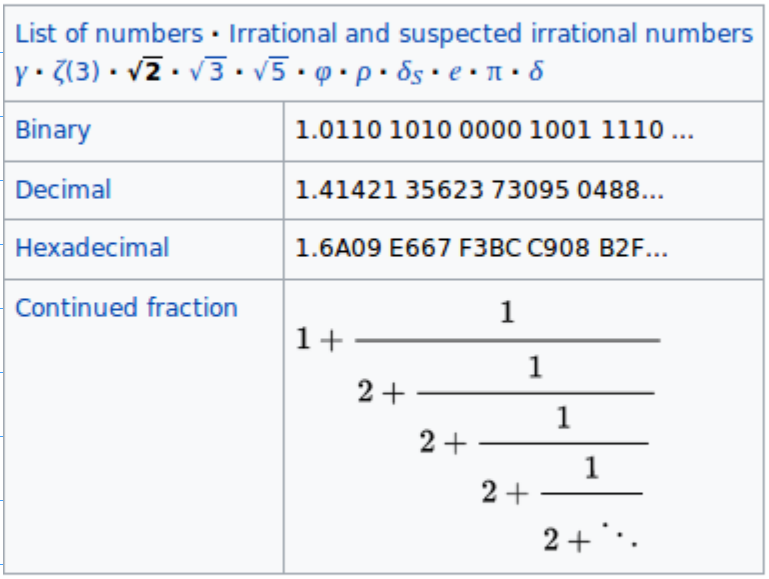


Type field [edit]

The Type field can be any of:

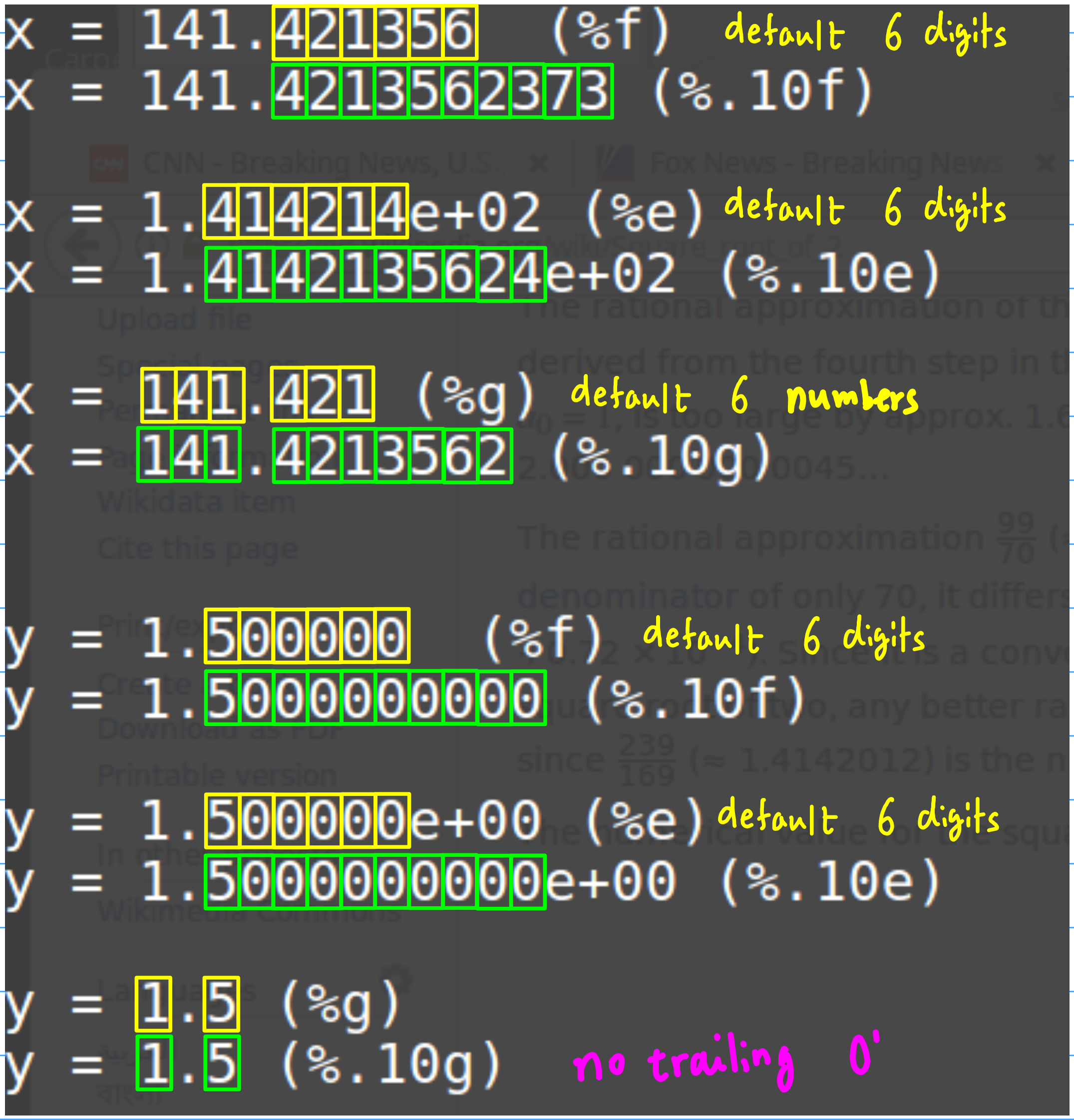
Character	Description
`%`	Prints a literal `%` character (this type doesn't accept any flags, width, precision, length fields).
`d`, `i`	`int` as a signed decimal number. `%d` and `%i` are synonymous for output, but are different when used with `scanf()` for input (where using `%i` will interpret a number as hexadecimal if it's preceded by `0x`, and octal if it's preceded by `0`.)
`u`	Print decimal `unsigned int`.
`f`, `F`	`double` in normal (fixed-point) notation. `f` and `F` only differs in how the strings for an infinite number or NaN are printed (`inf`, `infinity` and `nan` for `f`, `INF`, `INFINITY` and `NAN` for `F`).
`e`, `E`	`double` value in standard form ([`-`]d.ddd `e`[`+`/`-`]ddd). An `E` conversion uses the letter `E` (rather than `e`) to introduce the exponent. The exponent always contains at least two digits; if the value is zero, the exponent is `00`. In Windows, the exponent contains three digits by default, e.g. `1.5e002`, but this can be altered by Microsoft-specific `_set_output_format` function.
`g`, `G`	`double` in either normal or exponential notation, whichever is more appropriate for its magnitude. `g` uses lower-case letters, `G` uses upper-case letters. This type differs slightly from fixed-point notation in that insignificant zeroes to the right of the decimal point are not included. Also, the decimal point is not included on whole numbers.
`x`, `X`	`unsigned int` as a hexadecimal number. `x` uses lower-case letters and `X` uses upper-case.
`o`	`unsigned int` in octal.
`s`	null-terminated string.
`c`	`char` (character).
`p`	`void *` (pointer to void) in an implementation-defined format.
`a`, `A`	`double` in hexadecimal notation, starting with `0x` or `0X`. `a` uses lower-case letters, `A` uses upper-case letters.[3][4] (C++11 iostreams have a `hexfloat` that works the same).
`n`	Print nothing, but writes the number of characters successfully written so far into an integer pointer parameter. Note: This can be utilized in Uncontrolled format string exploits.

https://en.wikipedia.org/



List of numbers • Irrational and suspected irrational numbers γ • $\zeta(3)$ • $\sqrt{2}$ • $\sqrt{3}$ • $\sqrt{5}$ • φ • ρ • δ_S • e • π • δ	
Binary	1.0110 1010 0000 1001 1110 ...
Decimal	1.41421 35623 73095 0488...
Hexadecimal	1.6A09 E667 F3BC C908 B2F...
Continued fraction	$1+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\cfrac{1}{2+\ddots}}}}$

1.41421356237309504880168872420969807856967187537694807317667973799,

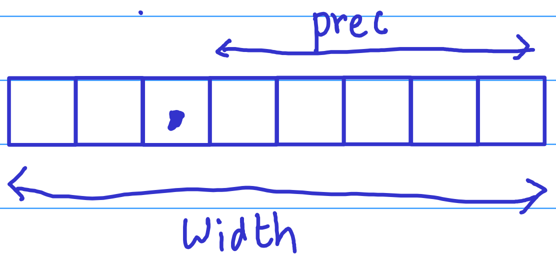


```
x = 141.421356   (%f)  default 6 digits
x = 141.4213562373 (%.10f)

x = 1.414214e+02 (%e) default 6 digits
x = 1.4142135624e+02 (%.10e)

x = 141.421  (%g) default 6 numbers
x = 141.4213562 (%.10g)


y = 1.500000   (%f)  default 6 digits
y = 1.5000000000 (%.10f)

y = 1.500000e+00 (%e) default 6 digits
y = 1.5000000000e+00 (%.10e)

y = 1.5 (%g)
y = 1.5 (%.10g)   no trailing 0'
```

https://en.wikipedia.org/

%[para][flags][width][.precision][length]type

-
+
sp
0
#



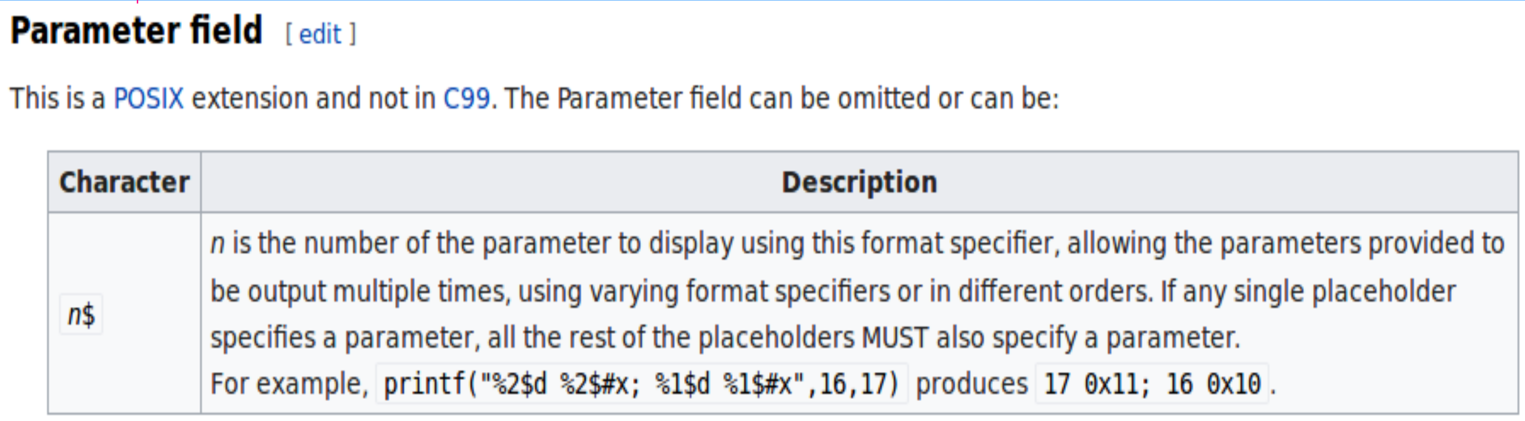
%f, %e

diff from %g

hh
h
l
ll
L
⋮

d
f
⋮

https://en.wikipedia.org/



Parameter field [edit]

This is a POSIX extension and not in C99. The Parameter field can be omitted or can be:

Character	Description
n$	*n* is the number of the parameter to display using this format specifier, allowing the parameters provided to be output multiple times, using varying format specifiers or in different orders. If any single placeholder specifies a parameter, all the rest of the placeholders MUST also specify a parameter. For example, `printf("%2$d %2$#x; %1$d %1$#x",16,17)` produces `17 0x11; 16 0x10`.

```
printf("%2$d %2$#x; %1$d %1$#x",16,17)
```

```
printf("%d %#x; %d %#x", 17, 17, 16, 16)
```

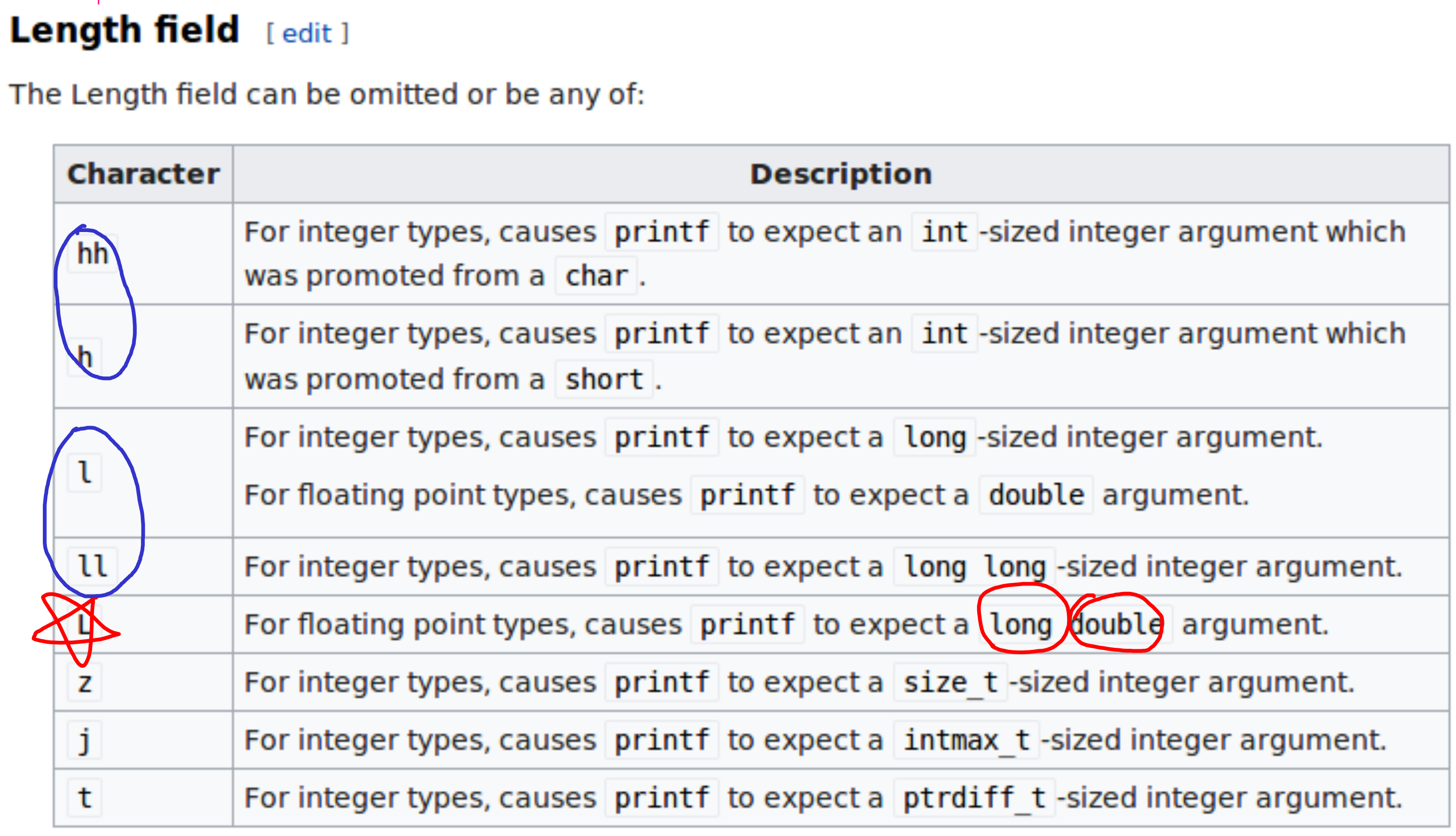
https://en.wikipedia.org/

Flags field [edit]

The Flags field can be zero or more (in any order) of:

Character	Description
`-` (minus)	Left-align the output of this placeholder. (The default is to right-align the output.)
`+` (plus)	Prepends a plus for positive signed-numeric types. positive = `+`, negative = `-`. (The default doesn't prepend anything in front of positive numbers.)
` ` (space)	Prepends a space for positive signed-numeric types. positive = ` `, negative = `-`. This flag is ignored if the `+` flag exists. (The default doesn't prepend anything in front of positive numbers.)
`0` (zero)	When the 'width' option is specified, prepends zeros for numeric types. (The default prepends spaces.) For example, `printf("%2X",3)` produces ` 3`, while `printf("%02X",3)` produces in `03`.
`#` (hash)	Alternate form: For `g` and `G` types, trailing zeros are not removed. For `f`, `F`, `e`, `E`, `g`, `G` types, the output always contains a decimal point. For `o`, `x`, `X` types, the text `0`, `0x`, `0X`, respectively, is prepended to non-zero numbers.

https://en.wikipedia.org/



Length field [edit]

The Length field can be omitted or be any of:

Character	Description
hh	For integer types, causes `printf` to expect an `int` -sized integer argument which was promoted from a `char` .
h	For integer types, causes `printf` to expect an `int` -sized integer argument which was promoted from a `short` .
l	For integer types, causes `printf` to expect a `long` -sized integer argument. For floating point types, causes `printf` to expect a `double` argument.
ll	For integer types, causes `printf` to expect a `long long` -sized integer argument.
L	For floating point types, causes `printf` to expect a `long double` argument.
z	For integer types, causes `printf` to expect a `size_t` -sized integer argument.
j	For integer types, causes `printf` to expect a `intmax_t` -sized integer argument.
t	For integer types, causes `printf` to expect a `ptrdiff_t` -sized integer argument.