E. Format Specifier Tricks

Here are a few tips and tricks for format specifiers (placeholders) in `printf` and `scanf` statements. This isn't an exhaustive list of the things you can do with format specifiers, but it includes the things you're likely to use most often. For complete information, see the excellent Wikipedia article on "printf format strings".

To start with, here's a list of format specifiers for some common variable types:

Format	Description
`%d`	Format for printing or reading an integer.
`%lf`	Format for printing or reading a `double`.
`%le`	Print a `double` in scientific notation.
`%lg`	Print a `double` in either scientific notation or normal notation, whichever is more appropriate.
`%c`	A single character.
`%s`	An array of characters (also called a "character string")
`%p`	A memory address (also called a "pointer").
`%u`	An unsigned integer.
`%ld`	A long integer.
`%lu`	An unsigned long integer
`%lld`	A long long integer.
`%llu`	An unsigned long long integer.
`%x`	A hexadecimal integer with lower-case letters.
`%X`	A hexadecimal integer with upper-case letters.
`%o`	An octal integer.

You can adjust the behavior of these format specifiers by adding modifiers to them. The following tables shows some tricks to help you make your program's output look just the way you want it.

Generic tricks

Example	Result	Description
`printf("%%")`	`%`	Print a literal `%` symbol.

Tricks for Integers

Example	Result	Description
printf("%**20**d",1234567890)	`          1234567890`	Print an integer, reserving a 20-digit-wide space for it. If the number isn't this long, add blank spaces on the left-hand side.
printf("%**-20**d",1234567890)	`1234567890`	The same as above, but add blank spaces on the *right*-hand side if necessary.
printf("%**8**d",1234567890)	`1234567890`	If the number won't fit in the specified width, use as much space as necessary.
printf("%**020**d",1234567890)	`00000000001234567890`	Pad the number with zeros on the left (if necessary) to make it 20 digits long.

Tricks for `doubles`

Example	Result	Description
printf("%**20**lf",M_PI)	`            3.141593`	Print a `double`, reserving enough space for 20 digits.
printf("%**5**lf",M_PI*1e8)	`314159265.358979`	If the number won't fit in the specified width, use as much space as necessary.
printf("%**20.10**lf",M_PI)	`        3.1415926536`	Reserve enough space for 20 digits (including the decimal point) and print 10 of those digits after the decimal point. Pad with spaces on the left-hand side if necessary.
printf("%**-20**lf",M_PI)	`3.141593`	As above, but pad on the *right*-hand side.
printf("%**020**lf",M_PI)	`0000000000003.141593`	Pad the number with zeros on the left (if necessary) to make it 20 digits long.

Tricks for Characters		
Example	Result	Description
`printf("%20c",'A')`	`                   A`	Print a character, reserving a 20-character-wide space for it. Fill the extra width with spaces on the left-hand side.
`printf("%-20c",'A')`	`A`	As above, but fill on the *right*-hand side.

Tricks for Strings		
Example	Result	Description
`printf("%20s","Testing")`	`             Testing`	Print a character string, reserving a 20-character-wide space for it. Fill the extra width, if any, with spaces on the left-hand side.
`printf("%-20s","Testing")`	`Testing`	As above, but fill with spaces on the *right*-hand side.
`printf("%4s","Testing")`	`Testing`	If the string won't fit in the specified width, use as much space as necessary.