Formatted I/O: printf

CSC 100 Introduction to programming in C/C++, Spring 2025

Marcus Birkenkrahe

February 9, 2025

README

- There is much more to `scanf` and `printf` than we've seen
- I/O is where the pedal hits the metal - where man meets machine
- In this notebook: conversion specifications for `printf`
- Practice workbooks, input files and PDF solution files in GitHub

printf

When it is called, `printf` must be supplied with:

1. a format string, like `"The output is:  %d\n"` (actually only `"%d"`)
2. any number of values to be inserted into the string at printing
3. the values can also be computed and assigned

Conversion specification

- A **conversion specification** is a placeholder like `d`
- Binary (machine) format is converted to printed (human) format
- General form: `%m.pX` where

	WHAT	EXAMPLE
m	minimum field width	`%4d` prints `123` as `_123`
p	precision after point	`%.3f` prints `3.141593` as `3.142`
X	conversion specifier	`d`, `e`, `f`, `g`

Examples:

```
printf("....|....|....|\n");

printf("%8d\n", 123); // print 123 on 8 places (right-aligned)

printf("%-8d\n", 123); // print 123 on 8 places (left-aligned)

printf("%10.3f\n", 3.141593); // print 3 decimals on 10 places (right)

printf("%-10.3f\n", 3.141593); // print 3 decimals on 10 places (left)
```

```
....|....|....|
     123
123
     3.142
3.142
```

Integer decimal %d or %i

- `d` or `i` display an integer in decimal (= base 10) form. `p` is the minimum number of digits to display the integer. Default is `p=1`.
- For example, the code below prints numbers with different precision values:
  - `%d` displays `int` in decimal form (minimum amount of space)
  - `%5d` displays `int` in decimal form using 5 characters
  - `%-5d` displays `int` on 5 characters, left-justified
  - `%5.3d` displays `int` on 5 characters, at least 3 digits

```
int i = 40;
printf("....|....|\n");
printf("%d\n",i); // decimal form (minimum amount of space)
printf("%5d\n",i); // decimal form using 5 characters
printf("%-5d\n",i); // on 5 characters, left-justified
printf("%5.4d\n",i); // on 5 characters, at least 3 digits
```

```
....|....|
```

```
40
   40
40
 0040
```

Floating point exponential %e or %E

- `e` displays a floating-point number in exponential ("scientific") notation, e.g. `10. * 10. * 10. = 1000. = 1.0e+03`.
- `p` indicates the digits after decimal point. If `p=0`, no decimal point is displayed.

```
printf("....|....|....|\n");
printf("%.E\n", 1.f);
printf("%-15.3E\n", 1000.f);
printf("%e\n", 1000000000000000.f);
printf("%15.e\n", 1000000000000000.f);
```

```
....|....|....|
1E+00
1.000E+03
1.000000e+15
          1e+15
```

Floating point fixed decimal %f

That's `f` as we already know it from many other examples. The precision p is defined as for `e`. Trailing zeroes are shown.

```
printf("....|....|\n");
printf("%10.3f\n", 100.1);
```

```
....|....|
   100.100
```

Variable floating point %g

- `g` displays a floating point number in either exponential format or fixed decimal format depending on the number's size.

- `p` is the maximum number of **significant** digits (**not** digits after the decimal point!) to be displayed.
- No trailing zeroes are shown. If there are no decimal digits after the decimal point, no decimal point is shown.
- How many lines and numbers are you expecting?

```
printf("%g\n%g\n%g\n", 299792458., 1.45e+03, 8000.);
```

```
2.99792e+08
1450
8000
```

- If you use `%g`, don't mess with the precision or the mantissa.

PRACTICE Printing with `printf`

- These exercises aren't going to be as much fun in OneCompiler as in Emacs. If you work in Emacs, you can fetch the practice file from here: tinyurl.com/printf-practice-org
- In OneCompiler, create a `NEW` file and add new sections with comments:

```
#include <stdio.h>

int main(void)
{
  // 1 Conversion specification
  printf("....|....|....|\n");
  ...
  ...
  // 2 Integer decimal %d
  printf("....|....|....|\n");
  ...
  ...
  // etc.
  return 0;
}
```

- You can open the exercises here on GitHub: tinyurl.com/printf-practice
- Upload your program URL to Canvas (`"In-class practice 6:  printf"`)

Conversion specification

Recreate the output below exactly, using only format specifiers (no extra white space).

```
: ....|....|....|
:   100100
: 200200
:         3.1416
: 3.141593
```

Solution:

```
printf("....|....|....|\n");
printf("%8d\n", 100100);
printf("%-10d\n", 200200);
printf("%13.4f\n", 3.141593);
printf("%-.6f\n", 3.141593);
```

```
....|....|....|
  100100
200200
       3.1416
3.141593
```

Integer decimal `d`

Show that the default for `d` is p=1. Print the numbers 1, 1, 100 and 10000 with the specifiers `%d`, `%.1d`, `%.5d`, `%.2d`. Print each expression on its own line, but use only ONE `printf` statement.

Solution:

```
printf("....|....|....|\n");
printf("%d\n%.1d\n%.5d\n%.2d\n", 1, 1, 100, 10000);
```

```
....|....|....|
1
1
00100
10000
```

Integer decimal precision `p`

Print the number 42 on a space of 10 characters with precision 5.

The result should look like this:

```
: ....|....|....|
:      00042
```

Solution:

```
printf("....|....|....|\n");
printf("%10.5d\n", 42);
```

```
....|....|....|
     00042
```

Scientific notation `e`

- Print 1, 1000.100, and 1,000,000,000,000,000 using `%e`.
- Provide for the required number of decimal positions (but not more)
- Print each expression on its own line with its own `printf` function.
- Add the header-argument `:results output` to the code block

Desired output:

```
: 1e+00
: 1.0001e+03
: 1e+15
```

Solution:

```
printf("%1.e\n", 1.);
printf("%.4e\n", 1000.1);
printf("%.e\n", 1000000000000000.);
```

```
1e+00
1.0001e+03
1e+15
```

Variable floating point `g`

- Use the format specifier `g` to display the following numbers: `200`, `3.142574654` with p=8, `2.71`, and `!5`.
- print each on a line of its own, but use only **one** `printf` statement to do it!
- `!N` is defined as the factorial of `N`.

Solution:

```
printf("....|....|\n");
printf("%g\n%.8g\n%g\n%g\n", 200., 3.142574654, 2.71, 5.*4.*3.*2.*1.);
```

```
....|....|
200
3.1425747
2.71
120
```