Width and Precision in printf()

When formatting output using the **printf()** function, **width** and **precision** are two important fields that allow us to control how the output is displayed on the screen.

> printf("%width.precision", args...)

Width Field

The width field specifies the minimum number of characters to be printed. If the output is smaller than the specified width, spaces or padding characters are added. The behaviour of width specifier depends on the type of data it is working on.

1. Default Behaviour

Without specifying width, the output is printed as it is.

2. Width with Integer

It specifies the minimum number of characters to be printed.

- If the number has fewer digits than the specified width, spaces (or zeros if using 0 flag) are added to the left by default.
- Negative sign (-) is counted as a character for width.

3. Width with Strings

It specifies the minimum number of characters for the string.

- If the string length is less than the width, spaces are added to the left by default.
- If the string length exceeds the width, the entire string is printed without truncation.

4. Width with Floating Point Numbers

Specifies the minimum number of characters for the entire floating-point value (including the decimal point and digits after it).

- If the value is shorter than the width, spaces (or zeros if using 0 flag) are added.

5. Dynamic Width

We can use a ***** to specify the width dynamically as a parameter to printf().

Example

C

```c
#include <stdio.h>
int main()
{
    int x = 124;
    printf("%d \n", x);
    printf("%5d \n", x);
    char y[] = "gfg";
    printf("%5s \n", y);
    float z = 1.2;
    printf("%5g \n", z);
    printf("%*s \n", 6, y);
    return 0;
}
```

Output

```
124
  124
  gfg
  1.2
   gfg
```

Flags with Width

Flags are provided with width to modify its normal behaviour. There are two flags for width specifier:

- **- (Left Align):** Aligns output to the left
- **0 (Zero Padding):** Pads with zeros instead of spaces (for numeric values).
- **+ (Positive Sign):** Prints positive sign for positive values.

Example

C

```c
#include <stdio.h>

int main()
{
    int x = 124;
    char y[] = "gfg";
    float z = 1.2;
    printf("%-5d %-5s %f\n", x, y, z);
    printf("%05d\n", x);
    printf("%+5d\n", x);
    return 0;
}
```

Output

```
124   gfg   1.200000
00124
 +124
```

Precision Field

The precision field is primarily used for floating-point numbers and strings. It controls the number of characters or decimal places to be printed. Like width, it can be dynamic and can be used along with precision.

The behaviour of precision specifier also depends on the type of data it is working on.

1. For Strings

Precision specifies the maximum number of characters to be printed.

- If the string has more characters than the precision, it is truncated.
- If the string has fewer characters, no truncation occurs.

C

```c
#include <stdio.h>

int main()
{
    char x[] = "geekforgeeks";
    printf("%.3s\n", x);
    printf("%5.3s\n", x);
    printf("%*.*s\n", 5, 3, x);
    printf("%-5.3s\n", x);
    return 0;
}
```

Output

```
gee
  gee
  gee
gee
```

2. For Floating Point Numbers

Precision determines the number of digits after the decimal point.

- If the number of digits exceeds the specified precision, rounding occurs.

%f: Prints the number in fixed-point notation.

C

```c
#include <stdio.h>

int main()
{
    double x = 1.345;
    printf("%.2f\n", x);
    printf("%5.2f\n", x);
    printf("%*.*f\n", 5, 2, x);
    printf("%0*.*f\n", 5, 2, x);
    printf("%-5.2f\n", x);
    return 0;
}
```

Output

```
1.34
 1.34
 1.34
01.34
1.34
```

%e: Prints the number in exponential notation with precision specifying the digits after the decimal.

C

```c
#include <stdio.h>

int main()
{
    double x = 1.345;
    printf("%.2e\n", x);
    printf("%5.2e\n", x);
    printf("%*.*e\n", 5, 2, x);
    printf("%0*.*e\n", 5, 2, x);
    printf("%-5.2e\n", x);
    return 0;
}
```

Output

```
1.34e+00
1.34e+00
1.34e+00
1.34e+00
1.34e+00
```

%g: Chooses between fixed-point or exponential form based on the value, with precision determining the significant digits.

C

```c
#include <stdio.h>

int main()
{
    double x = 1.345;
    printf("%.2g\n", x);
    printf("%5.2g\n", x);
    printf("%*.*g\n", 5, 2, x);
    printf("%0*.*g\n", 5, 2, x);
    printf("%-5.2g\n", x);
    return 0;
}
```

Output

```
1.3
  1.3
  1.3
001.3
1.3
```

3. For integers

Precision specifies the minimum number of digits to be printed. It basically has same meaning as width in this case except that it adds 0 in the beginning and does not consider the - sign as character.

Example

C

```c
#include <stdio.h>

int main()
{
    int x = 123;
    printf("%.5d\n", x);
    printf("%.5d\n", -x);
    printf("%5d\n", x);
    printf("%05d\n", -x);
    return 0;
}
```

Output

```
00123
-00123
  123
-0123
```