input or output. Here is a comprehensive table of the format specifiers used in C, categorized by the type of data they handle:

Common Format Specifiers in C

Specifier	Description	Example Usage	Explanation
`%c`	Character	`printf("%c", 'A');`	Outputs a single character.
`%d`	Signed decimal integer	`printf("%d", 42);`	Outputs a signed integer in base 10.
`%i`	Signed decimal integer (same as `%d`)	`printf("%i", -42);`	Outputs a signed integer in base 10.
`%u`	Unsigned decimal integer	`printf("%u", 42);`	Outputs an unsigned integer in base 10.
`%f`	Decimal floating-point number	`printf("%f", 3.14);`	Outputs a floating-point number in decimal notation (default precision is 6).
`%e`	Scientific notation (lowercase 'e')	`printf("%e", 3.14);`	Outputs a floating-point number in scientific notation.
`%E`	Scientific notation (uppercase 'E')	`print %E", 3.14);`	Outputs a floating-point number in scientific notation with 'E'.
`%g`	Use `%f` or `%e` based on the value	`printf("%g", 3.14);`	Outputs a floating- or `%f` format (whichever is shorter).
`%G`	Use `%f` or `%E` based on the value	`printf("%G", 3.14);`	Outputs a floating-point number, using `%E` or `%f` format (whichever is shorter).
`%x`	Unsigned hexadecimal integer (lowercase)	`printf("%x", 255);`	Outputs an unsigned integer in hexadecimal (base 16) using lowercase letters.
`%X`	Unsigned hexadecimal integer (uppercase)	`printf("%X", 255);`	Outputs an unsigned integer in hexadecimal (base 16) using uppercase letters.
`%o`	Unsigned octal integer	`printf("%o", 255);`	Outputs an unsigned integer in octal (base 8).
`%s`	String of characters	`printf("%s", "Hi");`	Outputs a string.
`%p`	Pointer address	`printf("%p", &var);`	Outputs a pointer address in implementation-defined format.
`%%`	Percent sign	`print %%");`	Outputs a literal percent sign.

Size Modifiers for Integer Types

Size modifiers are used in conjunction with the format specifiers for integers to specify the size of the data type:

Modifier	Specifier	Description	Example Usage	Explanation
`h`	`%hd`	Short signed integer	`printf("%hd", 32767);`	Outputs a short integer.
`h`	`%hu`	Short unsigned integer	`printf("%hu", 65535);`	Outputs a short unsigned integer.
`h`	`%hx`	Short unsigned hexadecimal integer	`printf("%hx", 255);`	Outputs a short unsigned integer in hexadecimal.
`h`	`%ho`	Short unsigned octal integer	`printf("%ho", 255);`	Outputs a short unsigned integer in octal.
`l`	`%ld`	Long signed integer	`printf("%ld", 123456789L);`	Outputs a long integer.
`l`	`%lu`	Long unsigned integer	`printf("%lu", 123456789UL);`	unsigned integer.
`l`	`%lx`	Long unsigned hexadecimal integer	`printf("%lx", 255UL);`	Outputs a long unsigned integer in hexadecimal.
`l`	`%lo`	Long unsigned octal integer	`printf("%lo", 255UL);`	Outputs a long unsigned integer in octal.
`ll`	`%lld`	Long long signed integer	`printf("%lld", 123456789012345LL);`	Outputs a long long integer.
`ll`	`%llu`	Long long unsigned integer	`printf("%llu", 123456789012345ULL);`	Outputs a long long unsigned integer.
`ll`	`%llx`	Long long unsigned hexadecimal integer	`printf("%llx", 255ULL);`	Outputs a long long unsigned integer in hexadecimal.
`ll`	`%llo`	Long long unsigned octal integer	`printf("%llo", 255ULL);`	octal.
`j`	`%jd`	intmax_t signed integer	`printf("%jd", (intmax_t)123456789012345LL);`	Outputs a maximum width integer.

`j`	`%ju`	uintmax_t unsigned integer	`printf("%ju", (uintmax_t)123456789012345ULL);`	Outputs a maximum width unsigned integer.
`z`	`%zd`	size_t signed integer (for signed interpretation)	`printf("%zd", (ssize_t)size);`	Outputs a size_t value (commonly used for size_t to be interpreted as signed).
`z`	`%zu`	size_t unsigned integer	`printf("%zu", size);`	Outputs a size_t value.
`t`	`%td`	ptrdiff_t signed integer	`printf("%td", (ptrdiff_t)ptrDiff);`	Outputs a ptrdiff_t value.
`t`	`%tu`	ptrdiff_t unsigned integer	`printf("%tu", (ptrdiff_t)ptrDiff);`	Outputs a ptrdiff_t value interpreted as unsigned.

Continue generating

Size Modifiers for Floating-Point Types

Modifier	Specifier	Description	Example Usage	Explanation
`L`	`%Lf`	Long double	`printf("%Lf", 3.14L);`	Outputs a long double floating-point number.

Width and Precision Specifiers

Width and precision specifiers control the minimum number of characters to be printed and the number of digits to appear after the decimal point for floating-point numbers.

Specifier	Description	Example Usage	Explanation
`%10d`	Minimum width of 10 characters	`printf("%10d", 42);`	Outputs the integer with a minimum width of 10, padding with spaces if necessary.
`%.2f`	Precision of 2 digits after the decimal point	`printf("%.2f", 3.14159);`	Outputs a floating digits after the dec
`%10.2f`	Minimum width of 10, with 2 digits after the decimal point	`printf("%10.2f", 3.14159);`	Combines width and precision specifiers. Outputs the number with a total width of 10 and 2 digits after the decimal point.

Continue generating

Example Usage

Here's a C program demonstrating various format specifiers:

c Copy code

```c
#include <stdio.h>

int main() {
    char ch = 'A';
    int num = 42;
    float pi = 3.14159;
    unsigned int unum = 255;
    char *str = "Hello, World!";
    long int largeNum = 123456789L;

    printf("Character: %c\n", ch);              // %c
    printf("Signed Integer: %d\n", num);        // %d
    printf("Unsigned Integer: %u\n", unum);     // %u
    printf("Float: %.2f\n", pi);                // %.2f
    printf("String: %s\n", str);                // %s
    printf("Hexadecimal: %x\n", unum);          // %x
    printf("Octal: %o\n", unum);                // %o
    printf("Pointer: %p\n", (void*)str);        // %p
    printf("Long Integer: %ld\n", largeNum);    // %ld

    return 0;
}
```

Continue generating

This program outputs various types using their respective format specifiers, showing how to format characters, integers, floating-point numbers, and more.

These format specifiers allow for

Continue generating

Continue generating

Upgrade plan
Get GPT-4, DALL·E, and more

Message ChatGPT

ChatGPT can make mistakes. Check important info.