NAME

printf, fprintf, sprintf − formatted output conversion

SYNOPSIS

```
#include <stdio.h>

printf (format [, arg ] ... )
char *format;

fprintf (stream, format [, arg ] ... )
FILE *stream;
char *format;

sprintf (s, format [, arg ] ... )
char *s, format;
```

DESCRIPTION

Printf places output on the standard output stream *stdout*. *Fprintf* places output on the named output *stream*. *Sprintf* places 'output' in the string *s*, followed by the character **\0**. The string *s* must be long enough.

Each of these functions converts, formats, and prints each *arg* under control of the *format*. The *format* is a character string which contains two types of objects: plain characters, which are simply copied to the output stream, and conversion specifications, each of which causes conversion and printing of the next successive *arg*.

Each conversion specification is introduced by the character %. After the %, the following appear in sequence:

- an optional minus sign − which specifies *left adjustment* of the converted value in the indicated field;
- an optional zero which specifies that zero-padding will be done instead of blank-padding;
- an optional digit string specifying a *field width;* if the converted value has fewer characters than the field width, it will be padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width;
- an optional period **.** which serves to separate the field width from the next digit string;
- an optional digit string specifying a *precision* which gives the number of digits to appear after the decimal point, for e- and f-conversion; the maximum number of significant figures, for g-conversion; or the maximum number of characters to be printed from a string; it also serves as a modifier in o- and x-conversion;
- an optional **l** or **h**, specifying that a following **d**, **i**, **o**, **x**, or **u** corresponds to a long integer (for **l**) or a short integer (for **h**) *arg*.
- a character which indicates the type of conversion to be applied.

A field width or precision may be * instead of a digit string. In this case an integer *arg* supplies the field width or precision. If the integer corresponding to a precision has the value −1, the effect is as if the precision and its preceding decimal point were both absent.

If the end of the *format* occurs between a % and its following format code, that entire format item is ignored.

The conversion characters and their meanings are:

d **i** **o** **x** **X**	The integer *arg* is converted to decimal (for either **d** or **i**),**octal,**or hexadecimal notation respectively. The letters **abcdef** are used for **x**- conversion, and the letters **ABCDEF** for **X**- conversion. If the *precision* is present, a single leading zero will be prepended to a non-zero value in o-conversion, and the string '0x' (or '0X') will be prepended to the value in x- (X-) conversion.
f	The float or double *arg* is converted to decimal notation in the style '[−]ddd.ddd' where the number of d's after the decimal point is equal to the precision specification for the argument. If the precision is missing, 6 digits are given; if the precision is explicitly 0, no digits and no decimal point are printed, unless left-justification and zero-padding are both specified, and the field width is strictly larger than the minimum required.
e **E**	The float or double *arg* is converted in the style '[−]d.ddde±dd' where there is one digit before the decimal point and the number of digits after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced. The **E** format code will produce a number with **E** instead of **e** introducing the exponent. If left-justification and zero-padding are both specified, any zeroes so generated will appear before the **e** (or **E**). If the precision is zero and no padding zeroes are generated on the right, no decimal point will appear.
g **G**	The float or double *arg* is printed in style **d**, in style **f**, or in style **e**(or **E** in the case of a **G** format code), whichever gives the requested precision in minimum space.
c	The character *arg* is printed if it is not **0**.
s	*Arg* is taken to be a string (character pointer) and characters from the string are printed until a null character or until the number of characters indicated by the precision specification is reached; however if the precision is missing all characters up to a null are printed.
u	The unsigned integer *arg* is converted to decimal and printed (the result will be in the range 0 to 65535 for integer values, or 0 to 4294967296 for long values).
%	Print a %; no argument is converted.

In no case does a non-existent or small field width cause truncation of a field; padding takes place only if the specified field width exceeds the actual width. Characters generated by *printf* are printed by calling *putchar*(3S).

EXAMPLES

To print a date and time in the form "Sunday, July 3, 10:02", where *weekday* and *month* are pointers to null-terminated strings:

```
printf("%s, %s %d, %02d:%02d", weekday, month, day, hour, min);
```

To print π to 5 decimals:

```
printf("pi = %.5f", 4*atan(1.0));
```

SEE ALSO

ecvt(3C), putchar(3S), scanf(3S), stdio(3S).

NOTES

For compatibility with earlier versions of *printf*, the format codes **D**, **O**, and **U** are currently implemented to mean the same as **ld**, **lo**, and **lu**. These usages should be avoided.

BUGS

Outrageous precision specifications on **e**, **f**, and **g** formats can cause failure.