NAME

printf, fprintf, sprintf, snprintf, vprintf, vfprintf, vsprintf, vsnprintf − formatted output conversion

SYNOPSIS

```
#include <sys/types.h>
#include <stdio.h>
#include <stdarg.h>

int printf(const char *format [, arg] ...);
int fprintf(FILE *stream, const char *format [, arg] ...);
int sprintf(char *s, const char *format [, arg] ...);
int snprintf(char *s, size_t n, const char *format [, arg] ...);
int vprintf(const char *format, va_list args);
int vfprintf(FILE *stream, const char *format, va_list args);
int vsprintf(char *s, const char *format, va_list args);
int vsnprintf(char *s, size_t n, const char *format, va_list args);
```

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'. **Snprintf** (Minix-vmd only) is like **sprintf** except that no more than *n*−1 characters are written to *s* followed by a '\0'.

The **v*printf** functions can be used to make functions like the first four by using the **stdarg**(3) method to process the argument.

Each of these functions converts, formats, and prints its arguments after the first under control of the first argument. The first argument 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 **%**. The remainder of the conversion specification includes in the following order

- Zero or more of following flags:
  - a '#' character specifying that the value should be converted to an ''alternate form''. For **c**, **d**, **s**, and **u** conversions, this option has no effect. For **o** conversions, the precision of the number is increased to force the first character of the output string to a zero. For **x**(**X**) conversion, a non-zero result has the string **0x**(**0X**) prepended to it. For **e**, **E**, **f**, **g**, and **G** conversions, the result will always contain a decimal point, even if no digits follow the point (normally, a decimal point only appears in the results of those conversions if a digit follows the decimal point). For **g** and **G** conversions, trailing zeros are not removed from the result as they would otherwise be.
  - a minus sign '−' which specifies *left adjustment* of the converted value in the indicated field;
  - a '+' character specifying that there should always be a sign placed before the number when using signed conversions.
  - a space specifying that a blank should be left before a positive number during a signed conversion. A '+' overrides a space if both are used.
- an optional digit string specifying a *field width;* if the converted value has fewer characters than the field width it will be blank-padded on the left (or right, if the left-adjustment indicator has been given) to make up the field width; if the field width begins with a zero, zero-padding will be done instead of blank-padding;
- an optional period '**.**' which serves to separate the field width from the next digit string;
- an optional digit string specifying a *precision* which specifies the number of digits to appear after the decimal point, for e- and f-conversion, or the maximum number of characters to be printed from a string;

- the character **l** specifying that a following **d**, **o**, **x**, or **u** corresponds to a long integer *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.

The conversion characters and their meanings are

dox The integer *arg* is converted to decimal, octal, or hexadecimal notation respectively.

X Like **x**, but use upper case instead of lower case.

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.

e The float or double *arg* is converted in the style '[−]d**.**ddd**e**±dd' where there is one digit before the decimal point and the number after is equal to the precision specification for the argument; when the precision is missing, 6 digits are produced.

g The float or double *arg* is printed in style **d**, in style **f**, or in style **e**, whichever gives full precision in minimum space.

c The character *arg* is printed.

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 0 or missing all characters up to a null are printed.

u The unsigned integer *arg* is converted to decimal and printed.

% 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 **putc**(3).

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

putc(3), **scanf**(3), **ecvt**(3), **stdarg**(3).