NAME

printf − formatted print

SYNOPSIS

printf(fmt, arg$_1$, ...);
char *fmt;

DESCRIPTION

Printf 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 argument to *printf.*

Each conversion specification is introduced by the character %. Following the %, there may be

- an optional minus sign "−" which specifies *left adjustment* of the converted argument in the indicated field;
- an optional digit string specifying a *field width*; if the converted argument 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; if the digit string is preceded with the character "0", the padding character will be the character "0". In this case the number is not interpreted as octal. If the digit string is not preceded with a zero, the padding character is the default character which is blank unless the "%>" option has previously been used.
- an optional period "." which serves to separate the field width from the next digit string;
- an optional digit string *(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;
- a character which indicates the type of conversion to be applied.

The conversion characters and their meanings are

d
o
x The integer argument is converted to decimal, octal, or hexadecimal notation respectively.

D
O
X The long integer argument is converted to decimal, octal, or hexadecimal notation respectively.

f The argument 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. The argument should be *float* or *double.*

e The argument is converted in the style "[−]d.ddde±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. The argument should be a *float* or *double* quantity.

c The argument character or character-pair is printed if non-null.

s The argument 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. If the string pointer itself is null, then the string "**(null)**" will be printed.

u

l The argument is taken to be an unsigned integer which is converted to decimal and printed (the result will be in the range 0 to 65535).

r The argument is taken to be the base address of a vector which contains a remote argument list. The first element in the list is a character string which replaces the current format. The remaining elements in the vector are accessed and converted as specified by the new format. A reversion to the original format will never occur; thus any characters in the original format following the "**%r**" will be ignored.

> The next character in the string "*fmt*" will replace the default padding character for the remainder of the string unless changed once more through the use of "**>**".

***** The next argument is taken to be a field width specification and is used accordingly. For example, "**%*d**" with *x* and *y* as the corresponding arguments in the argument list would be interpreted as specifying as decimal number *y* to be padded to a field width of *x*.

If no recognizable character appears after the %, that character is printed; thus % may be printed by use of the string %%. 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.

Printf is actually an interface to the C library *format* subroutine which performs all the necessary formatting. The *format* subroutine is called identically to printf with the exception of an additional argument preceding *printf*'s *fmt* argument. This new argument is the address of the subroutine to be called for every character of output generated by *format*.

If *printf* were written as a C subroutine it would thus appear as follows:

```
printf(fmt)
char *fmt;
{
extern putchar();

        return(format(putchar, "%r", &fmt));
}
```

SEE ALSO

putchar(3)

BUGS

Very wide fields (>128 characters) fail.

Format, (and consequently *printf*), is not recursive.